An Associative Search Network for Knowledge Acquisition: A VLSI Design Study

COINS Technical Report #83-04

Andrew Cromarty
Kurt Rudahl
Lynn Ruggles
Richard Sutton

Department of Computer & Information Science
University of Massachusetts
Amherst, MA 01003

An Associative Search Network for Knowledge Acquisition: A VLSI Design Study

COINS Techical Report #83-04

Andrew Cromarty Lynn Ruggles
Kurt Rudahl Richard Sutton

Department of Computer & Information Science
University of Massachusetts
Amherst, MA 01002

Abstract

An Associative Search Network (ASN) is a simple learning and information storage mechanism capable of acquiring, through interaction with its environment, a linear mapping from a set of input vectors (sensory stimuli) to a set of output vectors (actions). These devices have been used in the study of numerous non-trivial learning tasks, including landmark learning and control of a simulated robot arm and a simulated dynamic balancing problem.

As with other associative network structures, the operation of the ASN is characterized by fault tolerance and the ability to generalize, making it potentially useful for problem solving in a fashion suggestive of models of biological memory; it has, however, more general learning capabilities than other associative network devices. Unfortunately, ASN's also share with other associative network devices an inherently analog nature not easily implemented using current digital integrated circuit technology.

In this report, we describe an efficient, highly pipelined design for an NMOS VLSI implementation of an ASN consisting of several semi-independent elements each with 16 sensory inputs of 4-bit precision. Provisions are included for interface to a standard microprocessor for initializing, testing, and monitoring the chip. A limited degree of dynamic reconfiguration of the chip is possible, as a failsafe measure against on-chip component failure. Novel architectural structures developed in the course of this design are discussed.

We would like to thank Dr. Carton Foster and Dr. Andrew G. Barto for their assistance and helpful comments on this study. This report was funded in part by AFOSR and the Avionics Laboratory (Air Force Wright Aeronautical Laboratories) through contract F33615-80-C-1088. This Technical Report is COINS VLSI Working Paper #2.

TABLE OF CONTENTS

LIST OF FIGURES iii

Chapter

0. INTRODUCTION 2

Associative Search Networks: an overview 3
Rationale for a VLSI implementation 9
Design Overview 11

1. OVERVIEW OF THE NETWORK COMPONENT 12

Network Structure 12
Network Timing during an Iteration Step 16

2. THE COMBINING CIRCUITS 19

Adder Element 19
Adder Tree 20
Noise 22
Threshold 24
Shift Register 25
Reinforcement Signal 30

3. THE INTERSECTION ARITHMETIC PROCESSOR AND MEMORY UNIT 32

Reading, Writing, and Adding with Shift Registers 33
Distribution of Signals Among the Intersection Circuits . . 37
Multiplying with the "Shift-Adder" 38
Detailed Layout of Sub-components 43

4. MICROPROCESSOR INTERFACE 50

Overview 51
UPI implementation details 53

5. TIMING AND CONTROL FUNCTIONS 64

Overview 64
Operation 66
The Counter 67
The Comparator 67
The Decay Registers 68

6. SUMMARY 69

APPENDIX A. TIMING OF SIGNALS AND OPERATIONS IN ITERATE MODE 73

LIST OF FIGURES

0.1 An ASN interacting with its environment E. 6
1.1 Block Diagram of the ASN Chip Network Section 13
1.2 Floorplan Layout of the ASN Chip Network Section 15
1.3 Iteration cycle timing diagram 17
2.1 The Adder Tree 21
2.2 Noise element 23
2.3 Threshold Element 27
2.4 Shift register for input storage 28
2.5 X Input 29
2.6 Z Trace Element 31
3.1 Block diagram of the Intersection Circuit 34
3.2 Stick diagram of intersection circuit component wiring 35
3.3 3 bits of the Shift Adder 41
3.4 Logic & schematic diagram of the 1-bit adder circuit 45
3.5 Stick diagram of the 1-bit adder circuit 47
3.6 Logic for weight/trace shift registers (2 bits) 48
3.7 Stick figure of weight/trace shift register (2 bits) 49
4.1 Functional relationships of the UPI 54
4.2 Tri-state bidirectional pad circuit -- logic 55
4.3 Register -- 1 bit (typical) 56
4.4 Static edge-triggered flip-flop 57
4.5 Tri-state bi-directional pad circuit -- stick diagram 58
4.6 Stick figure for two bits of the static registers 60
4.7 Static edge-triggered counter (single flip-flop stage) 61
4.8 Counter decode circuitry -- approximate layout 62
5.1 The Timing FSA 65

0. INTRODUCTION

There has recently been increasing interest in the use of networks of simple processing elements, operating in parallel and communicating with one another by excitatory and inhibitory signals, as substrates for associative storage of knowledge [8, 9]. The most noteworthy feature of these associative memory networks is their use of distributed representations in which dispersed rather than localized patterns of activation encode information, which provides the network with the ability to generalize its stored associational patterns to other patterns according to the degree of similarity. This also implies a degree of built-in fault tolerance. Hinton [7] persuasively argues that this kind of representation, properly managed, has distinct advantages over conventional means of storing knowledge.

In this paper we present a VLSI design for one such form of network, an Associative Search Network (ASN). We shall consider briefly the formal underpinnings of ASNs, how they function, and why a VLSI implementation would be of interest. We shall then present in increasing levels of detail the layout of an NMOS chip which faithfully embodies a 16-input, 2-output ASN. Our design includes several significant architectural novelties, among which are:

- a pipelined bit-serial adder tree;
- a "shift-adder" to perform bit-serial multiplication;
- a self-contained 1-bit pipelined adder unit with carry, which carries to itself through a one-cycle delay;
- the reuse of structures (e.g. shift registers) for quite different purposes during different parts of the overall major clock cycle, with component behavior established by a select line fed by the timing Finite State Automaton (FSA); and
- the repetition and composition of simple functional units (such as one-bit adders) in a wide variety of applications, which greatly sped the design process by making it unnecessary to "reinvent the wheel" for each new design subtask.

Associative Search Networks: an overview

An Associative Search Network (ASN) is a cellular learning mechanism capable of acquiring, through interaction with its environment, a mapping from a set of input vectors to a set of output vectors, or actions [3, 4].[1] As in other associative memory networks, the mapping embodied by the ASN provides for generalization and fault-tolerance, properties which make associative memory networks both potentially useful for problem solving and suggestive of models of biological memory [8]. In particular, associative memory structures are characterized by

---

1. Such a device could be thought of as sampling various sources of sensory information in its environment and producing a response in accord with an internally stored map from its input sensors to its effectors.

1. Distributed knowledge storage,
2. Resistance to noise and damage, and
3. Ability to generalize, i.e. to produce a correct output when its sensory input data are similar to, but not exactly equal to, sensations it has previously encountered.

The ASN differs from many learning machines [1, 2, 9, 10, 12, 14, 15, 17, 18] in that it requires feedback only in the form of a scalar reward or punishment for correct or incorrect behavior, rather than requiring its external environment to provide correct error measures or training examples consisting of explicit desired input/output pairs. An ASN employs active generate-and-test search and reinforcement learning, rather than the more restrictive form of supervised learning employed by other associative memory networks. This more general learning capability greatly extends the range of problems in which associative memory networks can be useful.

The ASN combines two types of learning which are usually considered only separately. First, it solves a pattern recognition problem by learning to respond to each environmental stimulus, or "key", with an appropriate output pattern. This is the problem solved by the associative memory systems described in the literature; the method used is similar to stochastic approximation pattern recognition methods.[2]

---

2. See, for example, Duda and Hart [6] for a discussion of these techniques.

Second, the ASN simultaneously uses a different type of learning to determine which output pattern is in fact optimal for each key. It effectively performs a search using a stochastic automaton method to maximize a payoff or reinforcement function [13, 16].

Figure 0.1 shows a block diagram of an ASN interacting with an environment E. At each time step, the environment provides the ASN with two kinds of sensory data:

1. A vector X(t) representing n sensory inputs X(t) = $(x_1(t),\ldots,x_n(t))$, where each $x_i(t)$ is in the non-negative real interval [0,1].
2. A real-valued reinforcement or "payoff" signal z.

The ASN produces an output pattern $Y(t) = (y_1(t),\ldots,y_m(t))$, where each $y_i(t)$ is either a 0 or a 1; this output constitutes the action taken by the ASN. The action of the ASN influences the environment E.

One time step after taking some action (that is, after producing an output pattern), the ASN receives from E an evaluation of the appropriateness of that action for the situation in which the action was taken.[3] This evaluation takes the form of the payoff signal z. Note that the evaluation alone is not sufficient to determine whether the preceding action was the best possible in the given context. The associative search task is to learn, for each input vector, to perform the action which maximizes the payoff value. In other words, it must

---

3. A significant limitation of the ASN as formulated here is the assumption that evaluation is available immediately after an action is taken. This assumption will not hold in some problem domains; additional mechanisms may then be required to handle this problem.

The ASN receives a reinforcement or "payoff" signal z from the environment E and context signals $x_1(t),\ldots,x_n(t)$, and transmits actions to E via output signals $y_1(t),\ldots,y_m(t)$. The "output" from the environment serves as the sensory input to the ASN.

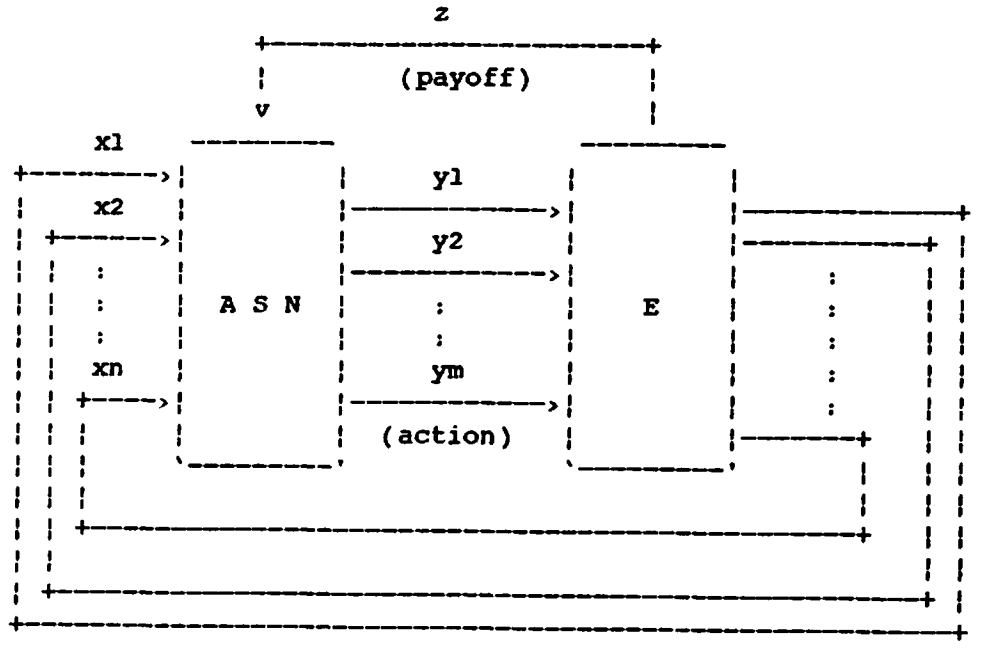


Figure 0.1: An ASN interacting with its environment E..

learn to perform the best action in every sensory situation; different actions can be optimal in different sensory contexts.

In simple ASNs such as the one implemented in this VLSI design study, a thresholded linear mapping from input X to output Y is learned. Each element i of the ASN sums up its n X-inputs, with each input $x_j(t)$ weighted by a real-valued memory variable $w_{ij}(t)$:

$$s_i(t) = \sum_{j=1}^{n} w_{ij}(t)\, x_j(t) \qquad (1)$$

These sums $S = (s_1(t),\ldots,s_m(t))$ then have some form of random noise added to them; this noise varies the ASN's behavior in order to effectively bring about search of its state space. These "noisy sums" are then compared with a threshold to yield the ASN's binary outputs $Y(t) = (y_1(t),\ldots,y_m(t))$:

$$y_i = \begin{cases} 1 & \text{if } s_i(t) + \text{NOISE}(t) > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

At each time step, each weight $w_{ij}(t)$, $i=1,..,m$, $j=1,...,n$, is updated according to the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + c\,[z(t)-z(t-1)]\,T_{ij}(t) \qquad (3)$$

where c is a positive fractional constant, and $T_{ij}(t)$ is another memory variable recording the extent to which the output and input corresponding to this weight have been simultaneously active in the recent past.[4]

---

4. Clearly, "recent past" will have different meanings in different problem settings; in general, we might expect it to be on the order of several seconds.

The idea here is quite simple. The assumption is made that the more recently an action was taken at the time of the change in the reinforcement (evaluation) signal z, the more likely the action is to have caused that change. Hence if an increase or decrease in z occurs, the ASN updates its events-to-actions map so as to take account of the evaluation provided by z, under the assumption that the change in z might have been due to some action recently taken by the ASN. The z signal thus plays the role of a scalar-valued correction to the (higher-dimensional) "sensorimotor" map.

The short term memory variable $T_{ij}(t)$ records which actions have recently been made and which stimuli were present when they were made. By the above weight change equation (equation 3), when an increase (decrease) in payoff z occurs, recent actions are made more (less) likely to occur in the presence of the inputs with which they recently co-occurred.

The short term memory variables $T_{ij}(t)$, i=1,m, j=1,n, each assume values in the positive real interval [0,1] and are updated according to:

$$T_{ij}(t) = (1-a)\,T_{ij}(t-1) + a\,y_i(t-1)x_j(t-1) \qquad (4)$$

where a, $0 \leq a < 1$, determines the decay rate of the short term memory trace. (We refer the interested reader to [3, 4] for a more detailed discussion of this form of learning rule.)

Rationale for a VLSI implementation

Digital simulations of analog processes have always been plagued by the fact that basic, natural analog functions force digital systems into quite unnatural acts (consider, for example, the relation "almost equal"). As a result, simulations tend to become heavily computation-intensive. Simulation of highly parallel networks of analog components (which are very often of interest) greatly compounds the problem, since it is difficult to simulate parallel, asynchronous processes on a sequential, synchronous machine, and harder still to be confident of the simulation's computational correctness.

Bergland (1981) argues in another context that requiring the structure of the solution to reflect the structure of the problem is one of the most effective methods for ensuring correctness (that is, appropriateness) of the solution. It is in this spirit that we have chosen to require the structure of the solution (as represented by the highly parallel architecture of the the ASN implementation) to reflect as closely as possible the structure of the underlying model system (the highly parallel abstract ASN). This approach provides some assurance that our solution adequately reflects the problem and that experience gained in designing and using this chip will provide useful insights into future ASN-chip designs.

We do not take lightly the very real concern that our simplification of the problem as presented here could be more efficiently solved by a general-purpose microprocessor. The benefit of a VLSI design for an ASN, apart from the philosophical one posed above, is that such a chip would serve as a precursor for similar highly parallel ASN designs which could be implemented on a much larger scale, where the performance of a serial processor, or indeed any imaginable network of them, would be quite poor indeed.

Finally, we must emphasize that this design report is a feasibility study of the VLSI implementation of such highly parallel associative structures, rather than a statement of our belief in the practical utility of the ASN as formulated here. Although capable of more general learning than other associative networks, the ASN we describe is, in our opinion, not a genuinely robust problem-solving system, but rather a simple example of a class of systems employing search and associative knowledge representation. As such it is a useful vehicle for exploration of the VLSI implementation of such systems.

Design Overview

The ASN chip consists of three primary sections:

1. The ASN elements themselves, each with 16 inputs producing one output (the design presently provides for two such elements per chip).
2. An interface to a standard microcomputer permitting direct examination and initialization of the ASN internal state.
3. An on-chip finite state machine acting as controller.

The ASN elements themselves can be considered in two parts: the intersection circuits, which perform computations on each input in parallel, and the combining circuits, which sum the contributions from each intersection circuit, add noise, threshold the resulting sum, and otherwise contribute to forming the element output.

1. OVERVIEW OF THE NETWORK COMPONENT

Network Structure

A block diagram of the network component of the design is shown in Figure 1.1. A chip floorplan diagram for the network component is shown in Figure 1.2.

The network component includes all machinery responsible for implementing the ASN with the exception of the generation of the control signals required by the network component during its operation; these are provided by the FSM and decay registers, and the registers maintained by the microprocessor interface. The network consists of two elements, corresponding to the two groups of structures running vertically in Fig. 1.1.

Each element receives the same 16 X-inputs (running horizontally through both elements) and a global reinforcement signal R_c computed from the z evaluation input, and produces a separate output signal $y_i(t)$. To each X-input corresponds a single pin of the external chip package, from which a 4-bit, unsigned bit-serial value is read which encodes positive sensor values between zero and one. The z-input is an 8-bit signed two's complement binary number read in bit-serially on a dedicated pin. The Y-outputs are single bit binary values; each directly drives a separate output pin. The weight and trace variables

evaluation, 8 bit, serial

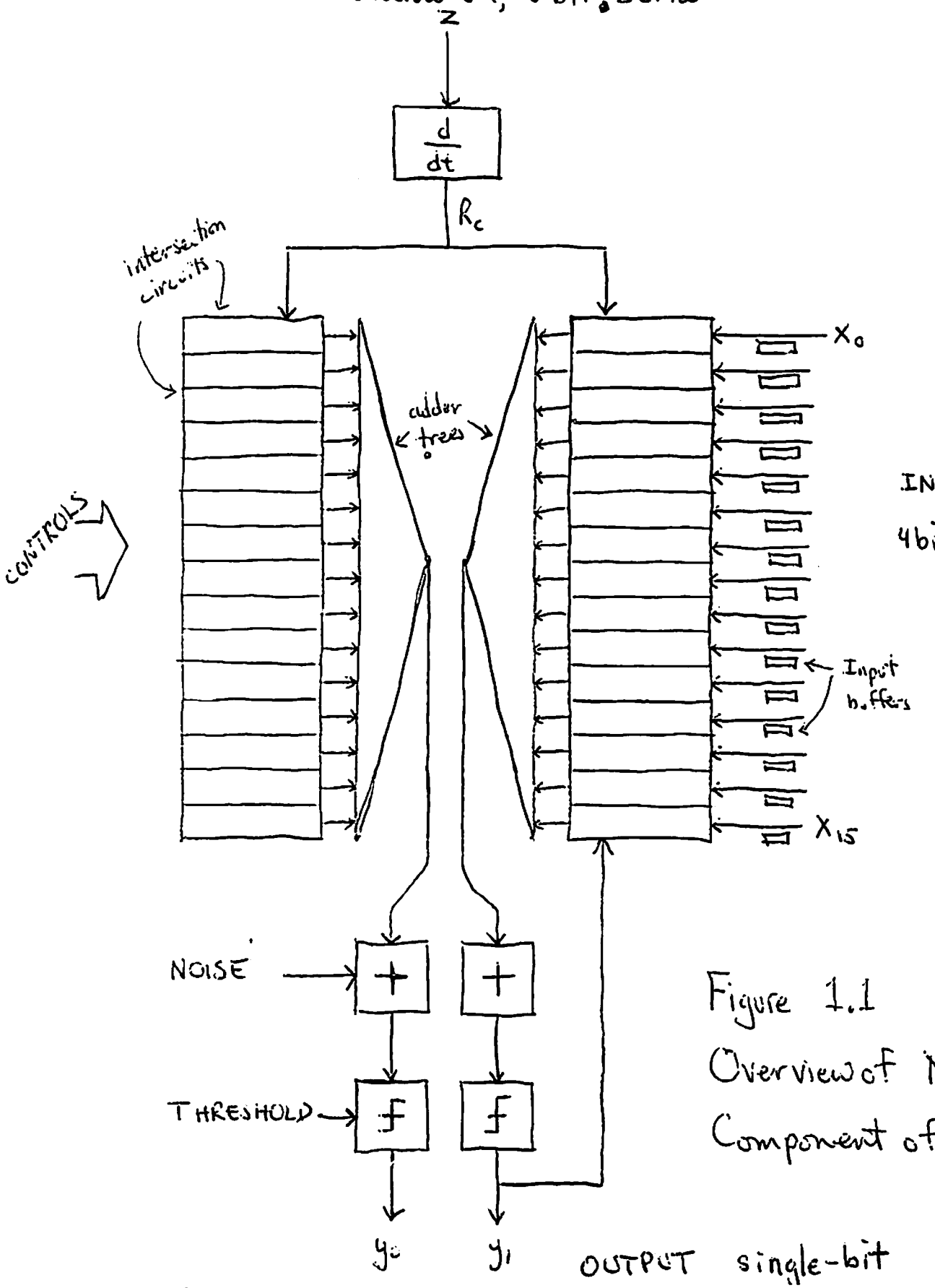


Figure 1.1
Overview of Network Component of Chip

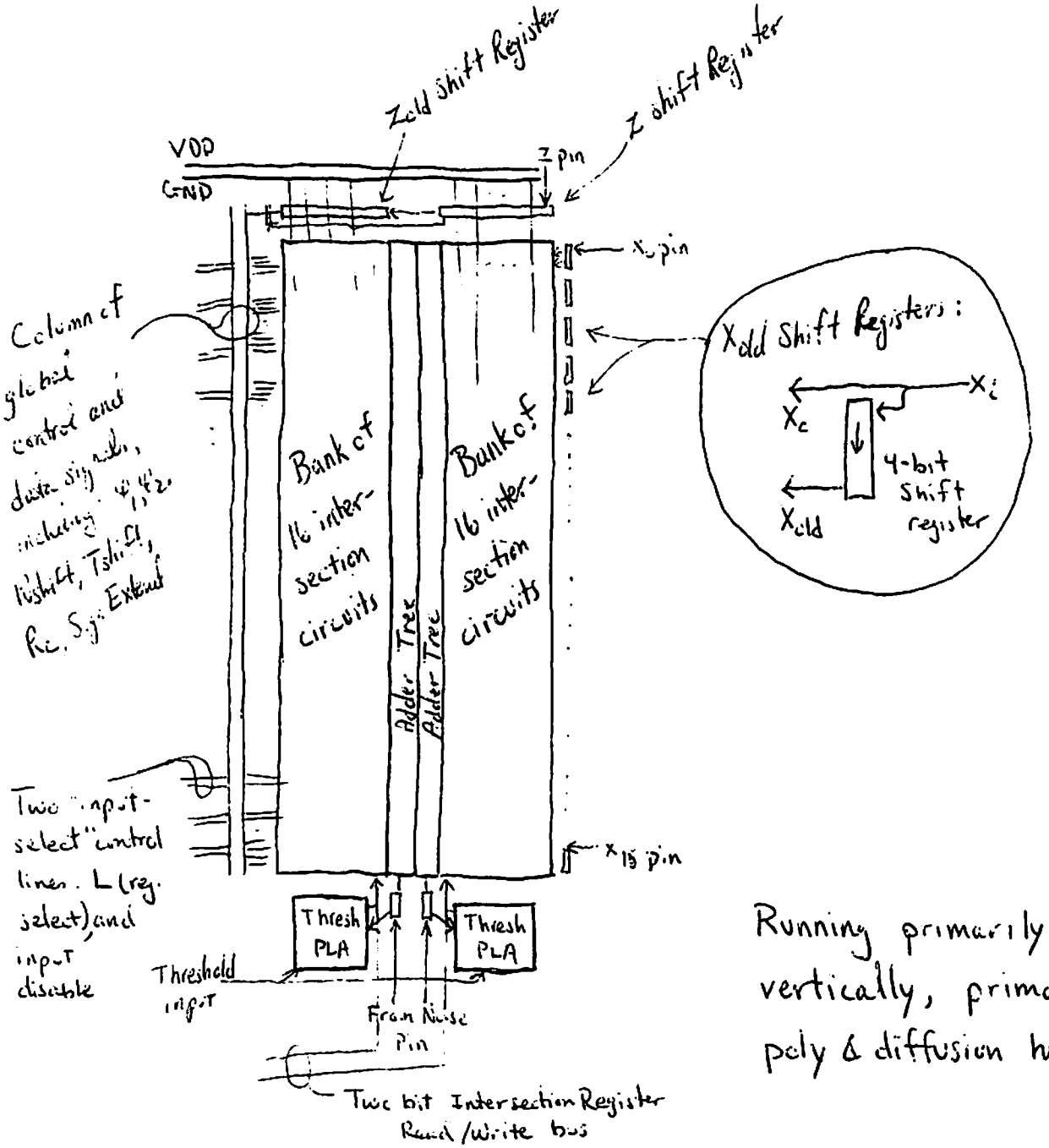
Figure 1.1: Block Diagram of the ASN Chip Network Section.

RD 12/13/91

Figure 1.2: Floorplan Layout of the ASN Chip Network Section.

The floorplan for the network section of the ASN chip. Control lines entering on the left are from the Finite State Machine (FSM) and the microprocessor interface components. Metal is running primarily vertically, poly and diffusion horizontally. The two largest vertical blocks are banks of 16 intersection circuits, with adder trees squeezed between them. A control and data bus runs vertically along the left side, gated by the two-input "input select" control lines that cross them. At the top below the power bus are the z and z_{old} shift registers; along the right edge are 16 X_{old} shift registers.

Figure 1.2: Floorplan Layout of the ASN Chip Network Section.



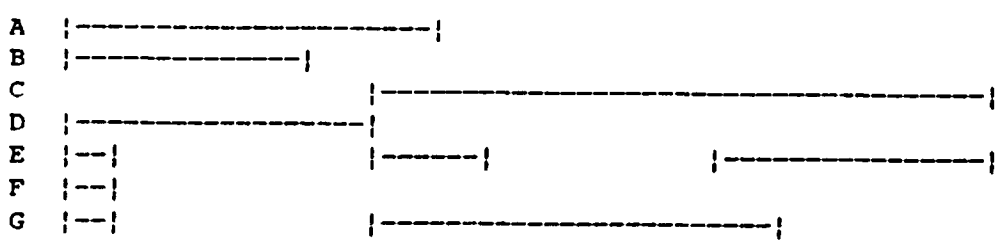
mentioned in section 0.1 are implemented as 16-bit registers located at each intersection between the 16 X-input lines and the 2 Y-output lines. These registers and their associated arithmetic machinery are labelled "intersection circuits" in figures 1.1 and 1.2 and are discussed in detail in section 3.

Network Timing during an Iteration Step

The general outline of processing during a network iteration is as follows: At the beginning of the iteration the X and z inputs are read into the chip simultaneously and stored in shift registers (shown along the top and right edge of the network component in figure 1.2). The inputs are multiplied by the weight registers at each intersection. The products of these multiplications are passed through the adder trees and the noise addition and threshold stages to produce the element Y-outputs. The input, multiplication, and addition processes are all pipelined and bit-serial.

While the result Y(t) is being computed, the output from the previous iteration is fed back into the intersection components along with a signal $R = z_c - z_{old}$ derived from the payoff input z, providing the basis for modification of the weight registers. The update of the trace register is overlapped with the product calculation mentioned above.

Condensed timing diagram of network operations during the iteration cycle:



A:	Output (Y) calculation	24 cycles
B:	Trace update	16-17 cycles
C:	Weight update	24-60 cycles
D:	Wx product calculation	20 cycles
E1:	Xc calculation	4 cycles
E2:	Rc .	8 cycles
E3:	Incremental weight update	16 cycles
F:	Xold calculation	4 cycles
G1:	Yold calculation	4 cycles
G2:	RT product calculation	24 cycles

Figure 1.3: Iteration cycle timing diagram.

Figure 1.3 is a condensed timing diagram for the iteration cycle. Periods A, B, and C -- output calculation, trace update, and weight update -- correspond exactly to equations 2, 3, and 4. The next period indicated in figure 1.3 (D) is that sub-period of the output calculation during which each intersection circuit is performing the multiplication of its weight by its X-input. Most of the other periods shown in this figure are those during which major data signals must be provided as input to the intersection circuit. The X, R_c, and X_{old} inputs are bit serial on dedicated input lines, while the Y_{old} input is a single bit held constant on the D data line during the period indicated.

Thus, one step in "iterate" mode (the normal operating mode of the chip) is between 45 and 61 cycles long, depending on the value of the Nc parameter, which is related to c in the weight update equation (equation 3) given in section 0.0. Appendix I details the time course of the iterate mode at the level of individual clock cycles.

2. THE COMBINING CIRCUITS

This section describes all parts of the network component except for the intersection circuit. This circuitry is responsible for:

- combining (adding) the outputs of the intersection circuits and forming the output of each element, and
- converting the network inputs (X and z) into a form appropriate for processing by the intersection circuits.

Adder Element

Each adder element adds two numbers together bit-serially, and produces a bit-serial output. The adder is provided with its input least significant bit (LSB) first. These two bits are added with the carry from the previous sum in a three-way addition operation, producing a one-bit sum as output from the adder element. The carry bit resulting from each addition is delayed one minor clock cycle and is then added into the next two incoming bits. The addition is complete when all bits from the two input sources have passed through the element. This adder element is used throughout the chip.

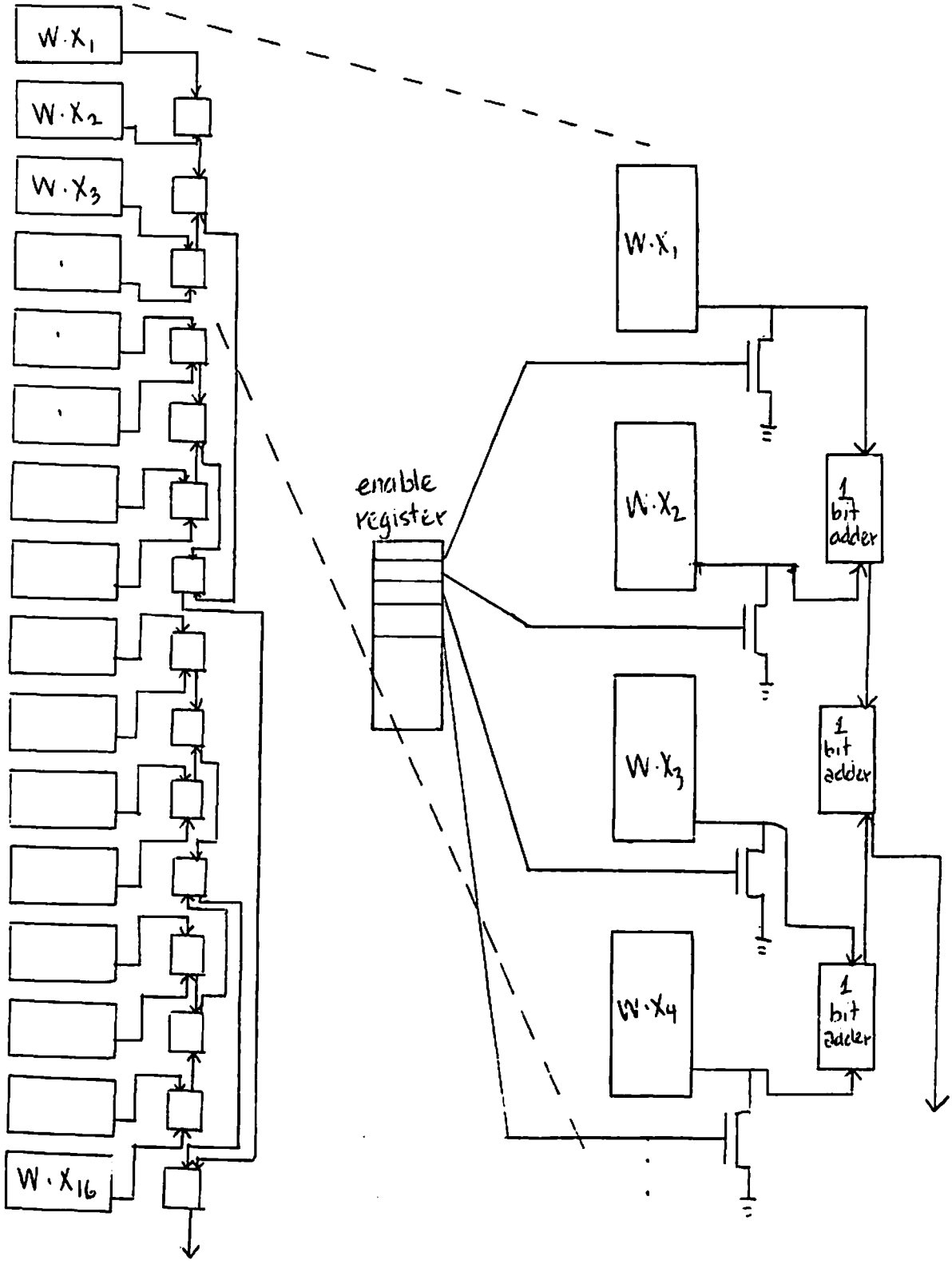
Adder Tree

After each intersection circuit has computed the product of each its corresponding weight and input, all of the products in each ASN element must be summed to calculate the result of equation (1). The addition of the intersection circuit products is performed by a pipelined adder tree. As the name implies, this component is a tree of adder elements (see figure 2.1).

The product calculated by each intersection circuit is transmitted bit-serially to the first level of adders in the tree. The least significant bit of each product output is added to the least significant bit of another product at the first level, forming the first sum. Each of these first sums is then added to another first sum, to form a second sum. The second sums are added to form third sums, and so forth. Thus, after the passage of four minor clock cycles, the tree has calculated the sum of all of the least significant bits of the products. At the end of 20 minor clock cycles the sum of all of the products will have been calculated by the tree.

The input to all adders in the first branch of the tree is gated with an enable line to allow the input to the adder to be sent either the product calculated by the intersection circuit or all zeros. This enable line is useful in case it is desired to ignore some of the inputs to the tree, a situation which might occur if there were defective elements on the chip. If an input is ignored, all inputs in the same position in the other elements on the chip will also be ignored, to

Figure 2.1: The Adder Tree.



maintain consistancy in the feedback calculated by the chip. If for some reason an input is to be ignored, its enable line is held high.

Sending all zeros through the adder element keeps the element in an empty or zero state until the weight information is available. This empty state guarantees that there will be no stray carry bits leftover from previous inputs.

The output from the adder tree is fed bit-serially into the noise addition circuit.

Noise

Each ASN element requires an independent source of random variation added into the sum computed by the adder tree in order to conduct a search among its possible actions. The noise source in this design is off-chip, read in through a single dedicated pin.

The noise element is a simple adder element (see figure 2.2). Its inputs are the bit-serial value output from the adder tree, and the noise value which is simply a random value input bit-serially from off chip to the noise element. To implement equation (2), each element on the chip must have the output of its adder tree added to a different noise value. To obtain distinct noise values for each element although we have only allowed for a single pin for the noise input line, the values input on this line will alternately go to the first and second element on the chip. A random 1 or 0 will be input on each phase of the

minor clock cycle. The bit for the first element will be saved and delayed one half minor clock cycle until the next bit, destined for the second element, has been sent on the second minor clock cycle phase. These two bits will then be added to their respective weight value during the next clock cycle, during which time the next two bits will be sent from off chip.[1]

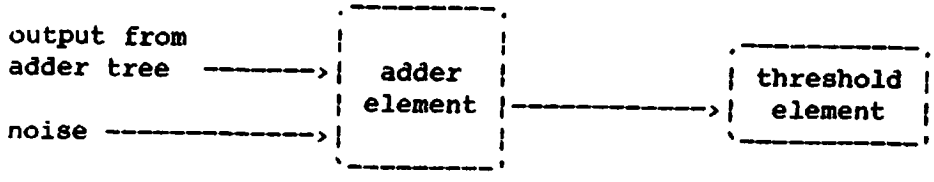


Figure 2.2: Noise element.

1. This scheme is adequate for the current VLSI implementation but will not scale up correctly for larger m (number of ASN elements). For larger ASNs it may prove desirable to include on-chip noise generation capabilities.

Threshold

Output from the associative search element is a binary value indicating that the sum $s_i(t)$ calculated by the element plus noise is higher or lower than a threshold value. This output value constitutes an input to the element during the next cycle of learning.

The threshold circuitry allows us to determines whether the element sum plus noise is greater than the output-threshold value which is read from off-chip when the chip is initialized. This value is stored in a shift register until it is needed for the threshold calculation. The value is then shifted out of the register, one bit at a time and is read by the threshold logic element and circulated to the most significant bit (MSB) of the register. After the threshold calculation has been performed and the value in the register circulated until it is back where it started, the threshold register stores the value until it is needed during the next iteration of the chip.

The threshold element has been implemented as a programmed logical array (PLA) (see figure 2.3). There are two bit-serial inputs to the array: the result from the noise calculation and the LSB of the threshold value. There is also a comparison-state value maintained during the threshold calculation. Initially this comparison-state value is zero. When the two values are input to the PLA, they are compared, and the following action is taken:

- If the bit in the element sum is larger, the state is set equal to 1;
- If the threshold value is larger, the state is reset; and
- If the two values are the same, the state value doesn't change.

This effectively compares the threshold value against the associative search element sum. After all the bits in the element sum and threshold value have passed through the threshold element, the state value is the value of y, which is fed back into the chip for the next weight calculation.

Shift Register

A shift register is used to hold values input from off chip until they are needed on the chip. In several cases, such as the X-input and the threshold input, these values will be used in either more than one step of the weight calculation or more than one iteration of the chip. In either case, we need to save the value until the appropriate signal from the FSA. This shift register is essentially identical to the stack implemented in [11], with the differences that the shift left was not needed and there is only one level to the stack (see figure 2.4).

The shift and circulate functions are performed bit by bit on only one word. To shift right, on phase one of the clock the Shift Right (SHR) line is held high and the input value passes into the inverter. On phase two, the Transfer Right (TRR) line is held high and the stored

Figure 2.3: Threshold Element.

Threshold element state diagram:

Threshold	Input	Current State	Next State
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

value passes through the next inverter. This datum is then passed to the next bit in the register at the next clock cycle. To circulate a value, on phase one the Transfer Left (TRL) line is held high, circulating the value in the register to the left. On phase two, the TRR line is held high, thus completing the circut.

In a few cases, the value in the shift register will be shifted out of the LSB and into an adder, while at the same time the LSB is also shifted around into the MSB of the shift register. In this way, the value in the shift register can be used, and also saved for future calculations without having to input the value again from off-chip.

As the X-input lines are read into the chip, the x value must be saved for a later calculation. This value is saved in a 4-bit shift register which will store the data until it is needed for the second calculation. When the x value for the next iteration is being read into the shift register, the old value of x is being shifted out the other end for input to the weight calculation. This element is shown in figure 2.5 and the insert in figure 1.2.

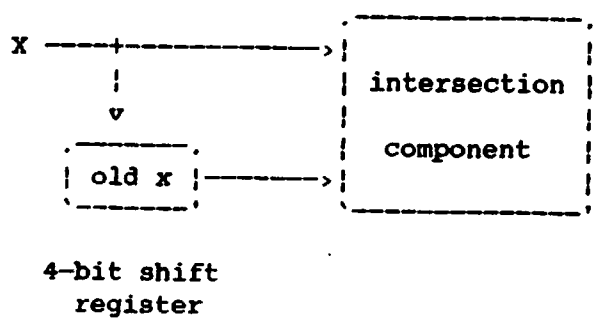


Figure 2.5: X Input.

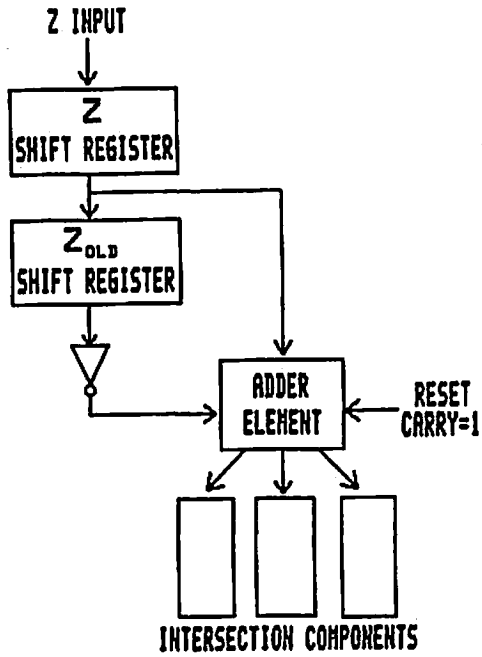
Reinforcement Signal

The ASN must learn to maximize a payoff or reinforcement signal z. This payoff signal is identical for every element on the chip. It is read at the same time as the X-input and saved in a shift register until it is needed (see figure 2.6).

Since the ASN is "seeking to maximize the payoff", so to speak, and since it is the net change in payoff over time rather than its absolute value that determines whether or not recent actions constitute an improvement, the difference between the current and previous values of the payoff signal are calculated and used to update the weight vector for each element. Thus if the payoff increases after an iteration, the ASN will tend to continue to perform the action that provided the increase, whereas if the payoff decreases, the element is more likely to do something else.

The payoff (z) is stored in a shift register. The previous value of the payoff (the value during the preceeding iteration of the chip -- z_{old}) is also saved, in a second shift register. The z value is added to the two's complement of the old z by clearing the adder, loading the carry bit with a 1 so that the first addition will also add 1 to the sum, and then adding the new z to the complement of the old z. This sum is sent bit-serially to the weight calculation on the R_c line. As the old z value is being shifted into the adder, the new z value is being shifted into both the adder and the old z register, thus becoming the old z for the next iteration.

Figure 2.6: Z Trace Element.



3. THE INTERSECTION ARITHMETIC PROCESSOR AND MEMORY UNIT

We now turn to the structure and operation of the "Intersection uit", which implements equations (3) and (4). The intersection circuit consists of the arithmetic and memory circuitry replicated at each intersection of the output and input lines of the network component, for the ultimate purpose of storing two 16-bit numbers and performing arithmetic operations with them and other signals that constitute the circuit's input. The intersection circuit is indeed the computational heart of the ASN chip.

At each intersection between the 16 input lines and the 2 output lines (one per ASN element) of the chip's network component are two 16-bit registers containing the network's short and long term memory variables. These registers are updated every major time step of the net (about once every 60 microseconds) in parallel, by arithmetic logic located at each intersection.

The intersection circuit performs all its arithmetic operations in pipelined, bit-serial fashion. These arithmetic operations include shifts, 16-bit by 8-bit signed multiplications and 16-bit signed additions; all these multiplications and additions must be performed simultaneously. Although the design of this circuit is surprisingly compact, the complexity necessary to perform these arithmetic and memory operations combined with the fact that this circuit must be replicated at each intersection make the area of the intersection circuit the

limiting factor for the overall design. (Note that the chip as a whole is space-limited, rather than time- or pin-limited).

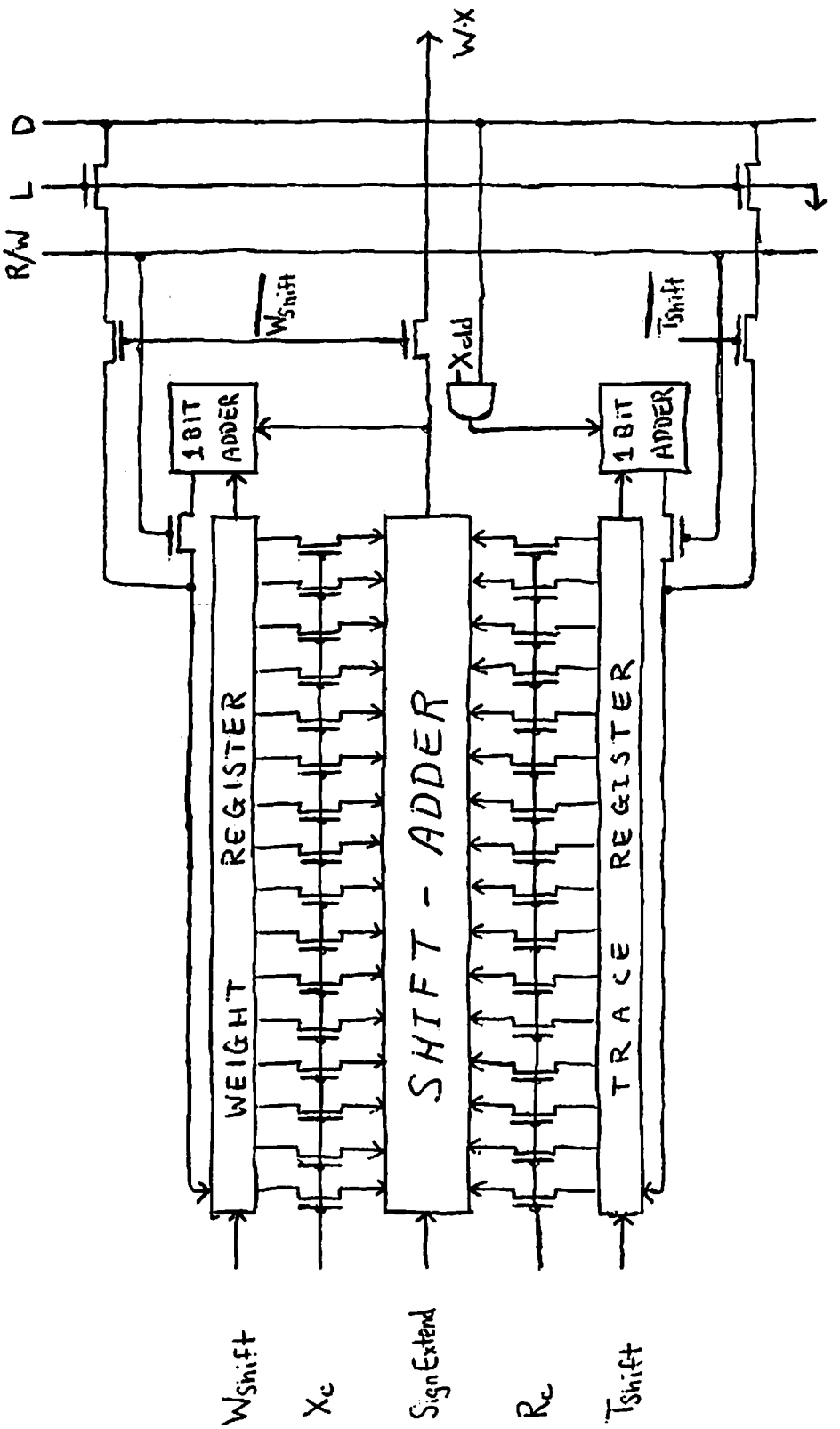
The logic of the intersection circuit is shown in Figure 3.1. This circuit is further broken down into subcomponents, with the interconnections between the subcomponents indicated in stick diagram notation in figure 3.2.

The centrally located "shift-adder" is responsible for performing the multiplications. We will consider it shortly.

Reading, Writing, and Adding with Shift Registers

Two 16-bit numbers are stored in the prominent 16-bit quasi-static shift registers labeled "Weight register" and "Trace register" in Figure 3.1 and Figure 3.2. These registers can be loaded with arbitrary values when the chip is in "pause" mode via the L, D, and R/W data and control lines. These lines can also be used to read the contents of the registers when the chip is in "pause" mode. The ability to read and write the registers in this way is not used during the normal or "iterate" mode of the chip; rather, these mechanisms provide access to the registers in a way that is largely independent of the correct functioning of the rest of the intersection circuit, for purposes of initializating, monitoring, and testing the chip.

Figure 3.1: Block diagram of the Intersection Circuit.



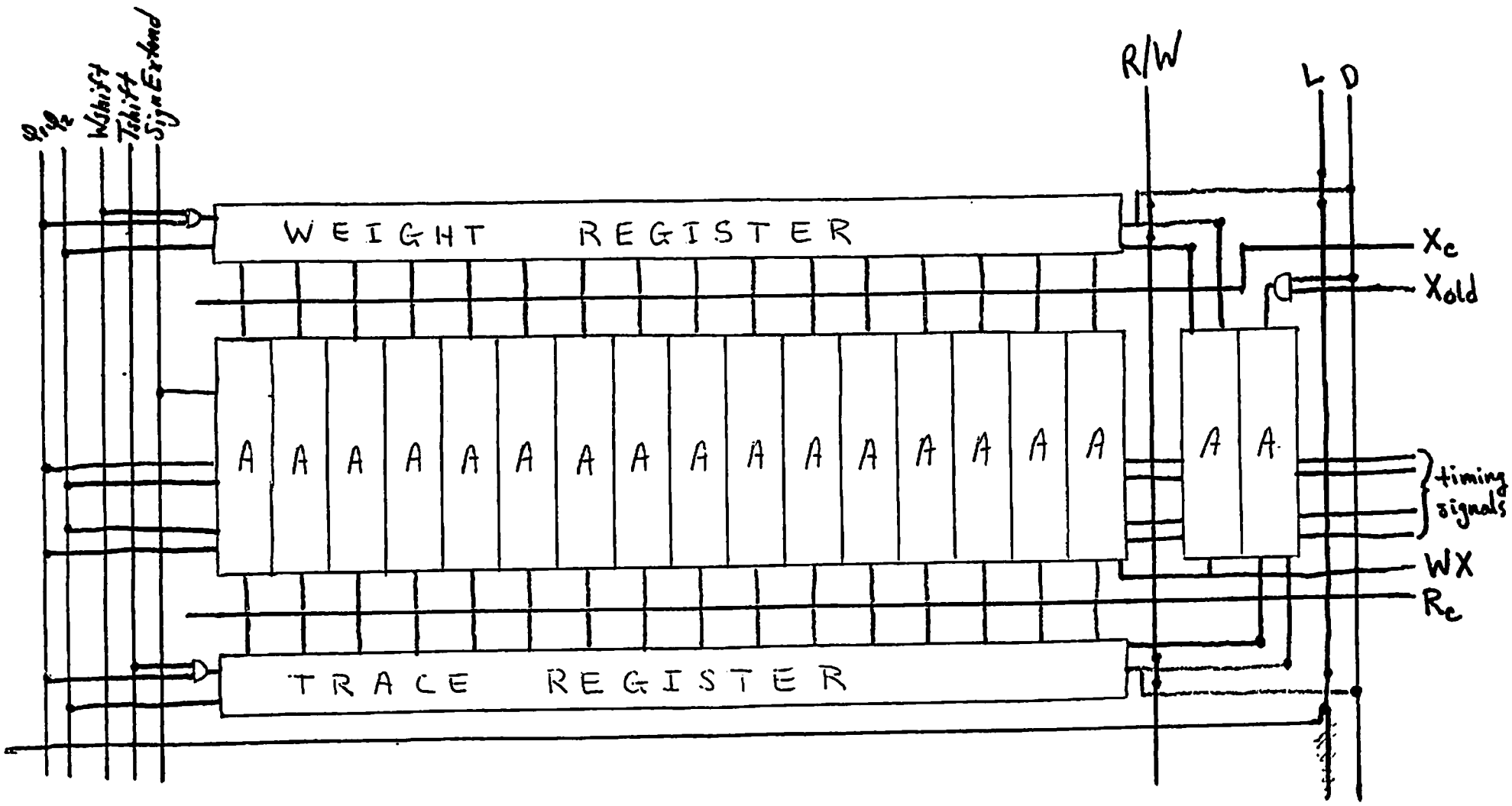


Figure 3.2: Stick diagram of intersection circuit component wiring.

Like all other operations performed by the intersection component, reading and writing of these registers occurs bit-serially. For example, the ASN chip can be instructed to dump the contents of a weight register: control lines are set so that the register performs a circular (unsigned) right shift, and the contents of the register are shifted out of the rightmost bit and onto the D data line. To transmit the register's contents, we "latch" (freeze) the inputs to the 1-bit adder at the end of each register so that the adder merely passes its input from the register directly through to the output unchanged. Similarly, to write new values into the register, the control lines are set so that the register performs a shift right, with the excess (right) bits being discarded and the new (left) bits being fed in from the D data line. Here the 1-bit adder again plays a trivial but indispensible role.

The 1-bit adders associated with each shift register are used when the intersection circuit adds a value to the current contents of one of the registers. The 16-bit datum to be added to the register must be provided bit-serially, least significant bit (LSB) first, to the 1-bit adder. As each bit of the input datum is provided, the register shifts right, and the LSB from the register is added to the other input to the 1-bit adder. The sum, provided by the adder, is loaded into the leftmost, or most significant, bit (the MSB) of the shift register , which has just been vacated. When the last two bits of the input have been added and the result stored in the leftmost bit, all the bits will have cycled around to their appropriate places.

Note that the 1-bit adder has no external carry source or destination. Because the addition is performed bit-serially, the carry bit of the addition can be simply retained from one addition to the next; if there is a carry, it is added to the next sum, which will form the next "more significant" bit. For bit-serial addition, the next sum is simply computed one time step later, by the same machinery. Thus, the carry is simply delayed by one clock cycle and added into the next bit of the sum. This sort of carry handling for bit-serial addition is used throughout this chip's design.

Distribution of Signals Among the Intersection Circuits

A word of clarification is in order regarding the distribution of the various control lines among the intersection circuits. Many control lines, such as Tshift, Wshift, R/W, and SignExtend, go to all 32 intersection circuits simultaneously and identically. Thus, all intersection circuits shift their weight and trace registers on the same clock pulse; their shifting cannot be individually controlled. Nearly all other control and data lines each go to two intersection components, one from each column, corresponding to those visited by the same input line. There are two exceptions to this. One is the major arithmetic output of the intersection circuit during "iterate" mode; there is one of these per intersection circuit. The other is the D data line, which is used for reading and writing the registers of each intersection

component during "pause" mode; there is one of these per column of intersection circuits contributing to the same output signal.

This distribution of the signals is essential to the manner in which the registers are read and written. In reading or writing, data are transferred bit-serially via the data line D. Two registers, one from each element, are always read or written at the same time, each via one of the two D lines. The two intersection circuits involved are selected via the L control lines, which run vertically accross the two elements. The register being written or read is determined by which register is shifting, which, in turn, is determined by the Wshift and Tshift global control lines. Finally, the R/W global control line determines whether a read or write operation is being performed (this control line must be held high during "iterate" mode).

Multiplying with the "Shift-Adder"

The shift-adder is a row of 16 1-bit adders, each with internal carry (as discussed above), and each connected one to the next as is shown in Figure 3.3. The shift register acts like an accumulator, i.e. we can add to its previous contents, and it also shifts the result right one bit each cycle (thus the name "shift-adder"). The intersection component uses the shift-adder to multiply the contents of the trace and weight registers with the current bit-serial input to the intersection circuit. Specifically, the bit-serial contents of the input line

labelled X_c in Figure 3.1 can be multiplied by the weight register, and the bit-serial contents of the input line labelled R_c can be multiplied by the trace register.

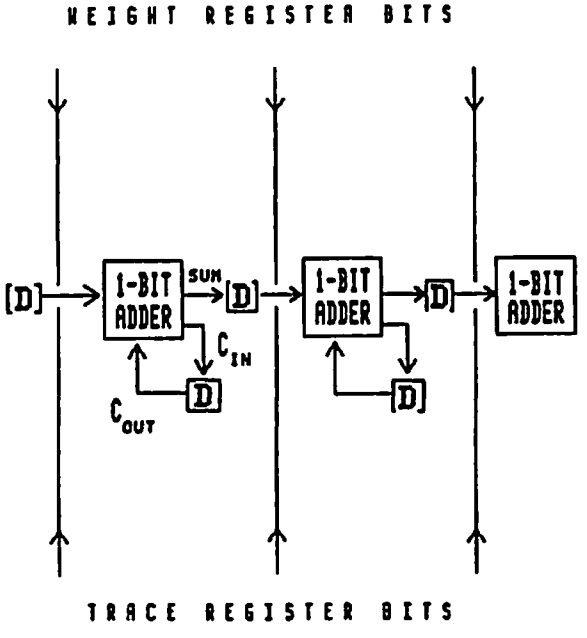
The principle underlying the design of the "shift-adder" is that binary multiplication can be performed by repeated additions and shifts. Consider multiplying the weight register by the bit-serial contents of the X_c input line. Assume the shift-adder is initially empty, i.e. all carry and shifting bits are zero. If the first bit of X_c is a one, then the contents of the weight register are added into the shift adder. If the second bit of X_c is also a one, then the contents of the weight register are again added into the shift-adder. However, before this is done, the shift-adder has shifted its contents one place to the right. This means that the second time, the weight register has been added in one bit higher, and has twice the significance, which is precisely what is desired for a multiplication. If a bit of X_c is not present, the weight register is not added in, but the shift-adder continues to shift, so that subsequent additions enter in at the appropriate level of significance. The right-shift operation performed by the shift-adder at each time step provides the bit-serial input to the next step in the pipeline. This simple technique performs multiplication correctly and in an exceedingly efficient manner.

If X_c is a 4-bit number and the weight register a 16-bit number, then their product will be a 20 bit number. As the shift-adder is only 16-bits long, the question arises as to what happens to the extra bits of the product, since there no place to save them while the rest of the product is being accumulated. The answer is that at each computational

Figure 3.3: 3 bits of the Shift Adder.

Note that this unit is constructed through composition of the 1-bit adder element.

Figure 3.3: 3 bits of the Shift Adder.



iteration of the shift-adder, the low order bit of the result has been completely computed and can thus be sent on as the bit-serial input to the next step of the pipeline.

Since the weight register contains a signed number and we are using two's complement representation for negative numbers, the shift adder must be able to perform a sign extend when it shifts out of the leftmost bit. The trace register, however, is unsigned, so the shift-adder must also be able to perform a shift without sign extend. The SignExtend control line determines whether or not the sign is extended.

Finally, note that nothing prevents one register from being multiplied with the help of the shift-adder while the other is performing an addition using its separate 1-bit adder as described previously. This, in fact, happens most of the time during the operation of the intersection circuit. For example, in one case the trace register is multiplied by the R_C input and the result (after dropping some low-order bits) is added into the weight register as it is shifted out of the shift adder.

Detailed Layout of Sub-components

Figure 3.4 shows the logic of the 1-bit adder circuit used throughout this design, and Figure 3.5 shows the corresponding stick diagram for two shift-adder designs, one employing random logic and the other "structured" function-block logic with the requisite driver circuitry. Because of its large replication factor in the intersection circuits and hence the extent of its contribution to the overall size of the chip, this adder circuit was highly optimized for space; in fact, two versions were designed, one using "random logic" and one employing "function block" logic in the spirit of "structured" design techniques, as depicted in 3.5.

Figure 3.6 shows the logic of 2 bits of the storage and shift register used for weights and traces in the intersection circuits, and Figure 3.7 shows the corresponding stick diagram. A special shift register is used here so that it interfaces properly with the space-optimized 1-bit adders. One of these shift registers runs just below and one just above the row of 16 1-bit adders in the shift-adder.

Figure 3.4: Logic & schematic diagram of the 1-bit adder circuit.

Logic of the 1-bit adder circuit:

A	B	Cin	G	K	GorK	Cout	SUM
0	0	0	0	1	1	0	0
0	0	1	0	1	1	0	1
0	1	0	0	0	0	0	1
0	1	1	0	0	0	1	1
1	0	0	0	0	0	0	1
1	0	1	0	0	0	1	0
1	1	0	1	0	1	1	0
1	1	1	1	0	1	1	1

Figure 3.4: Logic & schematic diagram of the 1-bit adder circuit.

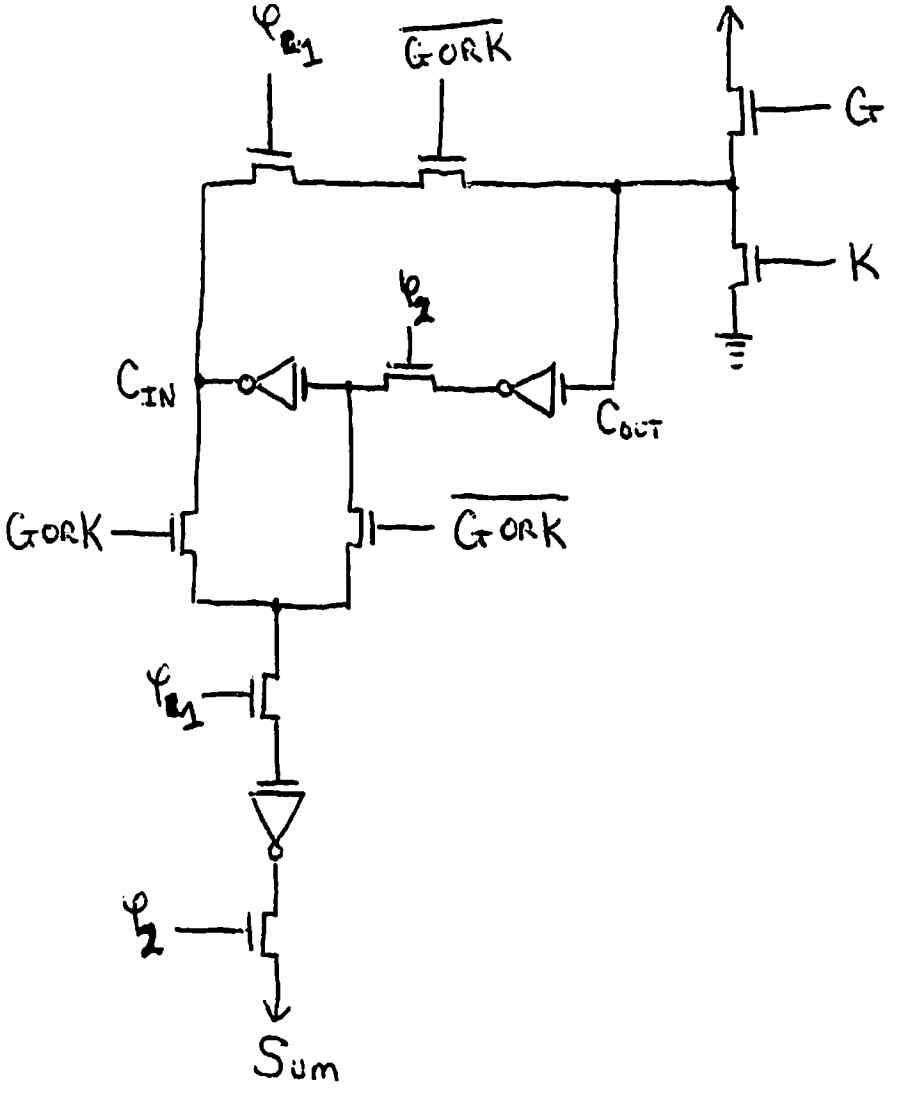


Figure 3.5: Stick diagram of the 1-bit adder circuit.

1-bit adder with delayed retained carry and delayed sum output. (a) Random logic implementation. (b) Function-block design. The seven diffusion columns of the function block correspond respectively (left-to-right) to: A&B, A, B, B, A, A&B, and ~A&~B (i.e. zero). Columns 2 and 3 provide A|B, columns 4 and 5 AxorB, and columns 6 and 7 AeqvB. The logic table for the block is:

		C out	S
C in	High	A|B	A eqv B
	Low	A&B	A xor B

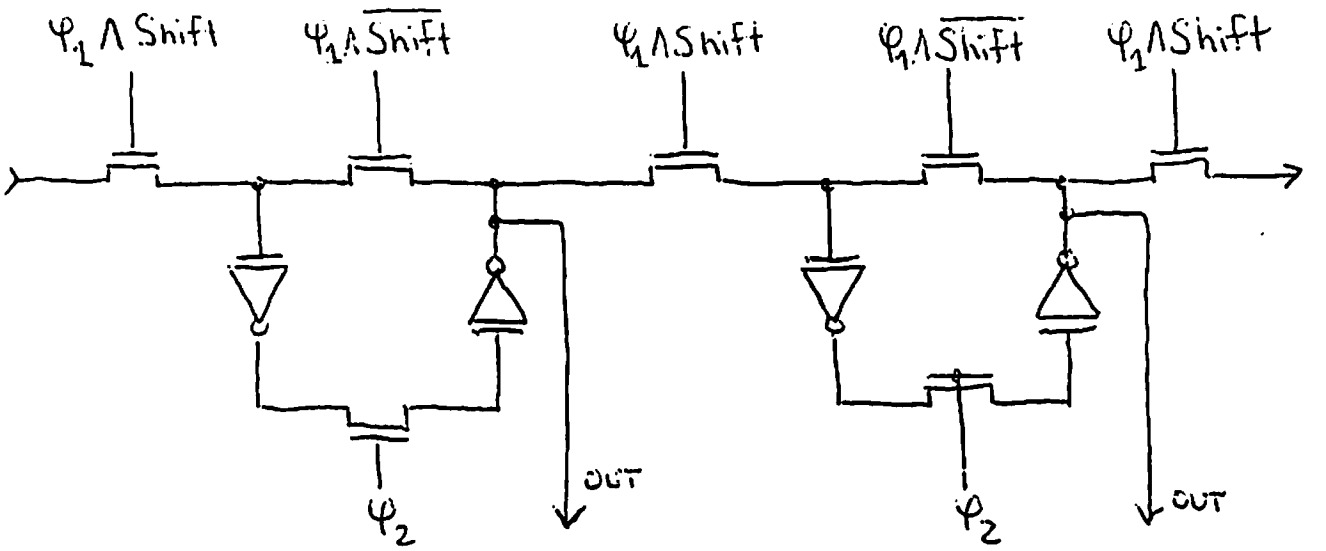


Figure 3.6: Logic for weight/trace shift registers (2 bits).

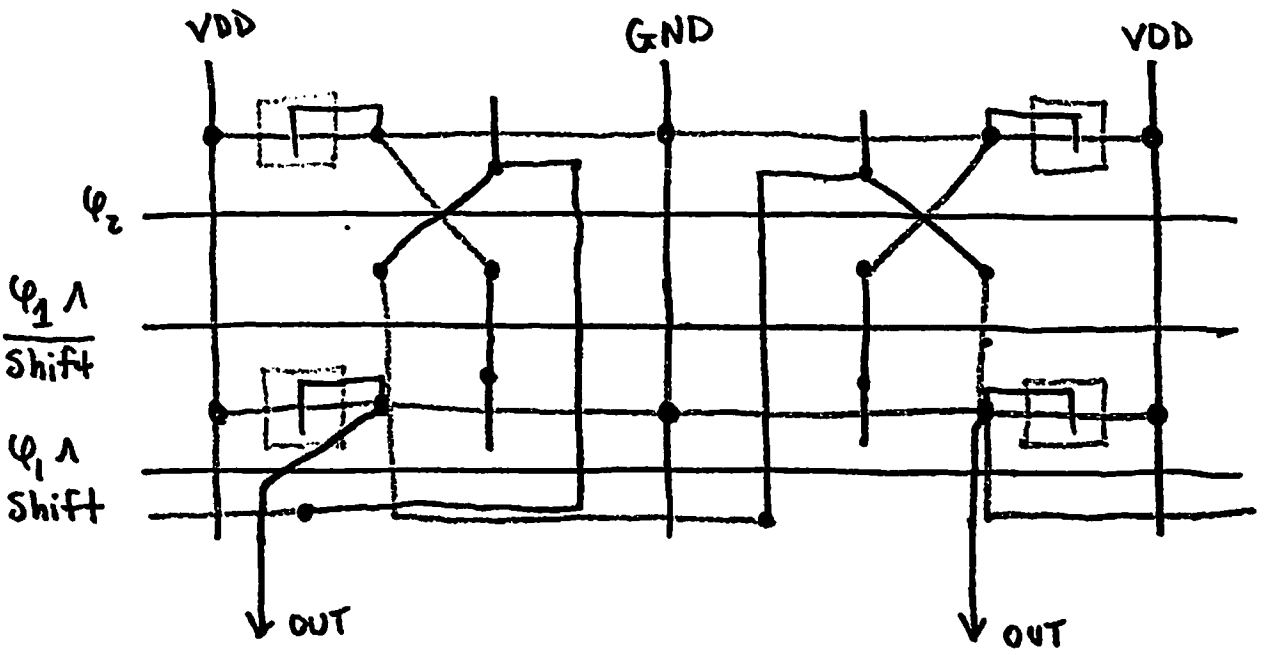


Figure 3.7: Stick figure of weight/trace shift register (2 bits).

4. MICROPROCESSOR INTERFACE

An interface to a standard microprocessor bus has been provided to permit both read and write access to all important registers of the chip. When the chip is functioning normally in a research environment, the microprocessor interface (UPI) permits initialization of the chip to a known condition and examinination of its internal state to see how rapidly the register values are converging, how many iterations have occured, etc. Likewise, the current state may be saved and used to reinitialize the chip at some later time. Further, a single chip can be given a great deal of experience solving a particular problem, and then its state can be read out and copied into many "naive" chips so they can all benefit from the first's experience.

Of more immediate concern to us in our roles as VLSI designers, however, this interface provides great flexibility in testing, debugging and to a limited degree reconfiguring the prototype chip. To this end, the UPI has been made to the greatest extent possible independent of the other functioning units of the chip. The price we pay for this independence, if we are also to avoid excessive silicon real estate, is that the data appear on the microprocessor bus in a rather scrambled form; the microprocessor itself will bear the burden of unscrambling the bits and converting them to a more useable form.

Overview

The UPI has been designed to conform to a common microprocessor, the 6502 family; however, the ASN chip can be interfaced to any microprocessor or could be run without a microprocessor at all. The interface consists of eight bidirectional tri-state data lines, three control lines, and two pins for the two-phase clock which is also used by all other circuits on the chip and defines the minor clock cycle. The three control lines consist of: CHIP SELECT, WRITE ENABLE, and PAUSE; all are active-low.

The registers to which we most require access are the weight and trace registers; there are 16 of each of these per element. As discussed above, the chip is presently being designed for two elements for reasons of space limitations, although all design decisions have been made on the basis of 4 or more, so that in the distant future when this becomes an amazing commercial success we will not be limited in our expansion plans. Two elements implies a total of 64 16-bit registers, plus a few others to be discussed, thus suggesting a minimum of 8 address lines. We have selected instead a scheme which requires no address lines but depends rather on a precise sequence of register access. Each assertion of the CHIP SELECT (while in PAUSE mode) toggles a counter which is decoded to select the access. A decoded counter was chosen over a long shift register because we are accessing a 3-D rather than linear array; it also avoids the problem of extra bits creeping into the shift register. Note that since address generation is

independent of the WRITE SELECT line, each access can be a read or write in any sequence.

The final configuration combines a minimum of chip geometry overhead with a maximum of inconvenience for the microprocessor: each register in the intersection circuits is read/written bit-serially using the register's inherent shift-register capability, with one bit (corresponding to each element's output) appearing in parallel on the data lines at a given instant. Since we are presently designing for only two elements, this means that only two of the eight data lines are actually being used for this access; we have retained the full eight to allow for future expansion. The counters and other remaining registers are read/written 8-bit-parallel.

The other registers to which access is provided are:

1. a Status 'Register' (currently 3-bits: the PAUSE mode flag plus the two element outputs). This is not actually a register, but a direct reading of the internal lines.

2. a 16-bit Enable Register, one bit per network input, which permits disabling the contribution of any input to allow for chip defects.

3. the 8-bit C Register. Contains N_C.

4. an 8-bit Control Register of which one bit per element will be used to select whether that output is bit-serial or a single thresholded binary value. An additional bit disables the addition of noise into the output computation.

5. a 16-bit Iteration Counter which counts the number of iterations the chip has executed. Writing to either half of this register clears both halves. If space on the chip permits, the Iteration Counter will be expanded to 24 bits.

UPI implementation details

The UPI consists primarily of the tri-state bus driver/receivers and some (static) ripple counters and registers; Figure 4.1 shows the general functional interrelationships of the UPI, although the actual circuit layout is somewhat different with (for example) the NAND gates flagged with '( )' being subsumed into the decoder circuit.

Notice that although both "phi-1" and "phi-2" clock inputs are shown, the UPI itself uses only phi-2 and its complement, thus permitting phi-1 to be adjusted as required by other parts of the chip without regard to parameters of the external microprocessor bus. The constraints on the two clock signals are:

1. Phi-2 is identical to the microprocessor phi-2 signal and should be in the range of (approx) 100 kHz to 2 mHz.
2. Phi-1 may be unrelated to the microprocessor signals except that it must not overlap phi-2.

The bus drivers are adapted from Mead and Conway [11, p165]: schematics and stick diagrams of a single bit are shown in Figures 4.2 and 4.5, respectively. Because the chip is operating synchronously with the microprocessor bus, the output latch specified by Mead and Conway is not necessary and has been deleted. The 6502 specifies that write data is available only at the leading edge of phi-2, so the data input circuit has been replaced by a dynamic edge-triggered register as shown.

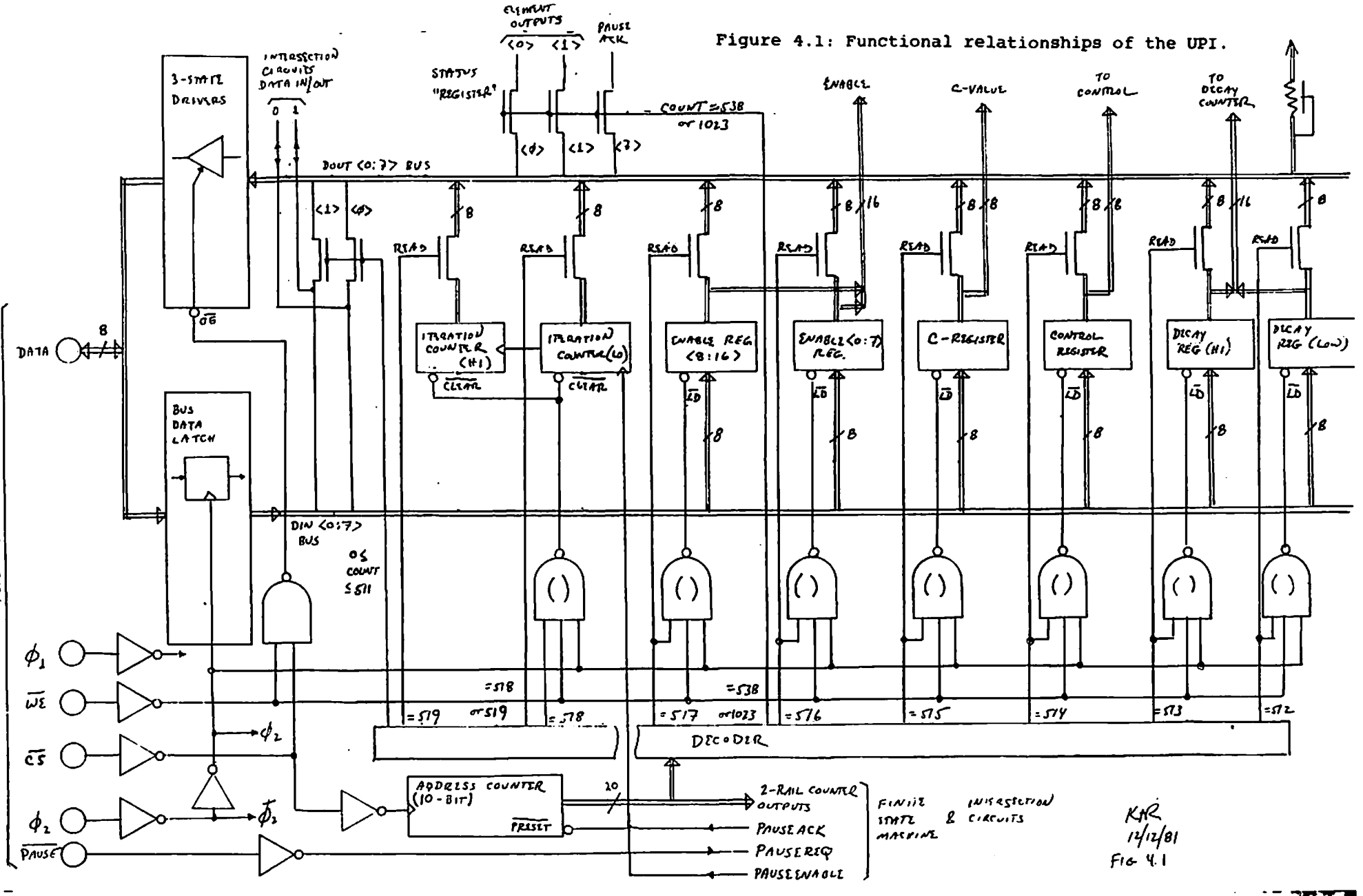


Figure 4.1: Functional relationships of the UPI.

KHR
12/12/81
Fig 4.1

Figure 4.2: Tri-state bidirectional pad circuit -- logic.

(See Mead and Conway [11], p.165 and plate 16.)

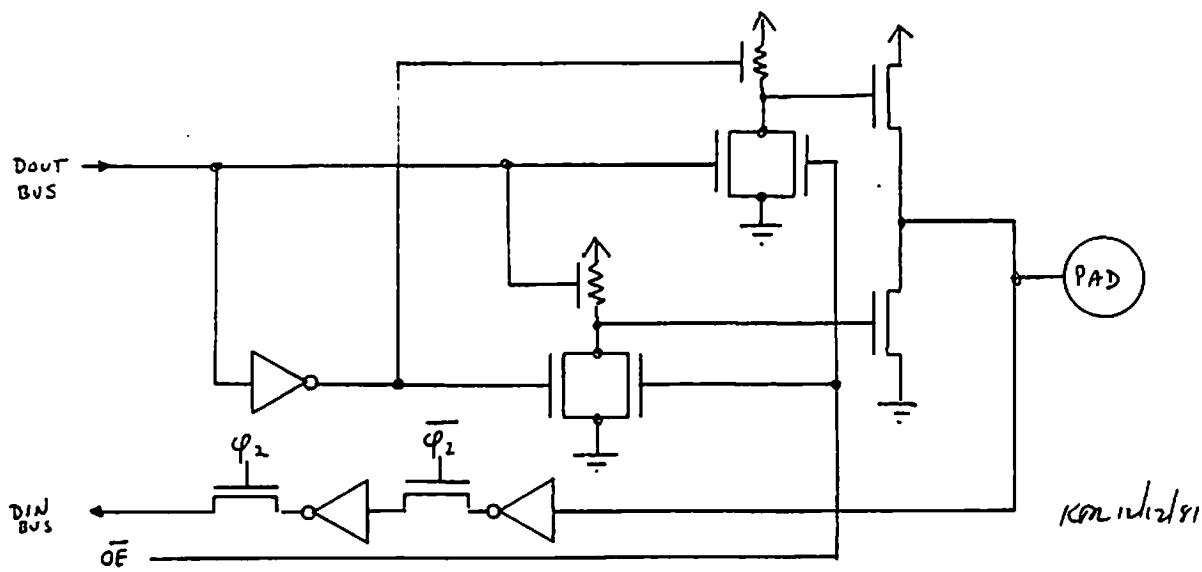


Figure 4.3: Register — 1 bit (typical).

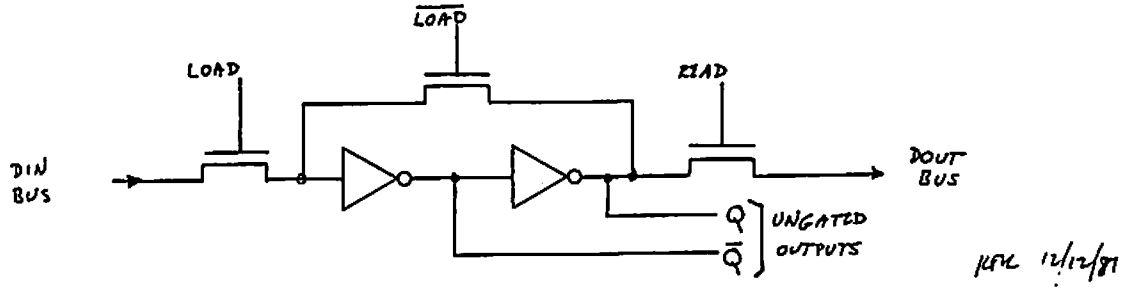


Figure 4.4 Static edge-triggered flip-flop.

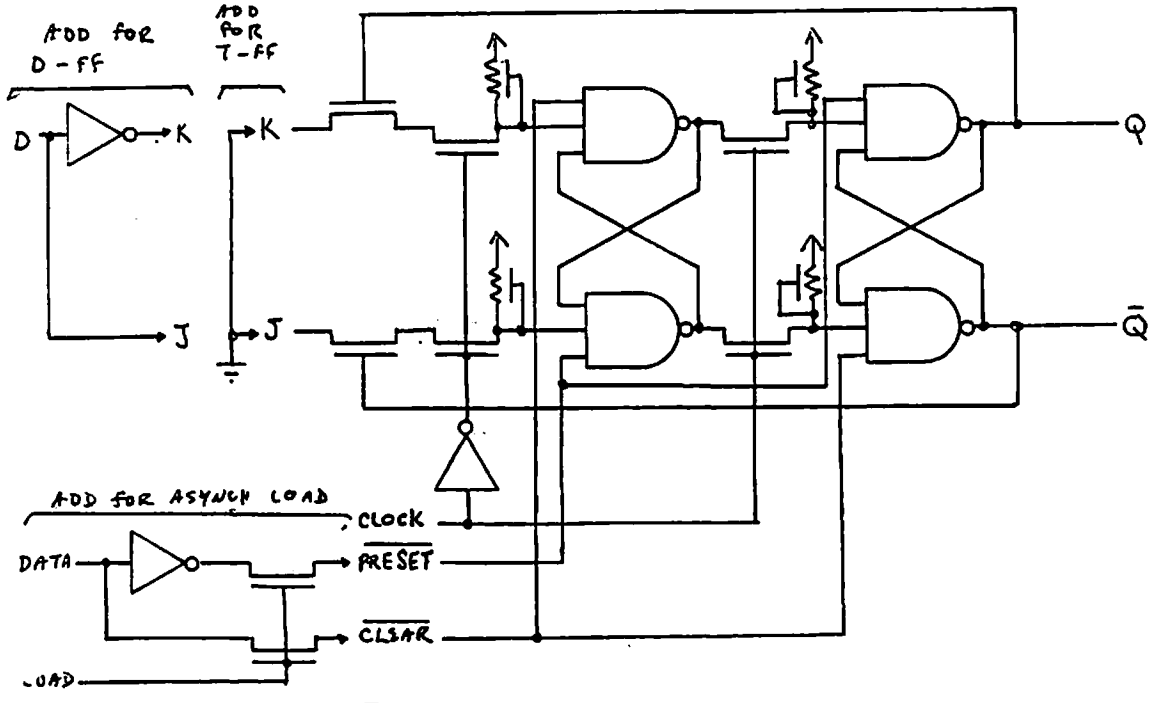


FIG. 4.4 STATIC EDGE-TRIGGERED FLIP-FLOP

Figure 4.5: Tri-state bi-directional pad circuit -- stick diagram.

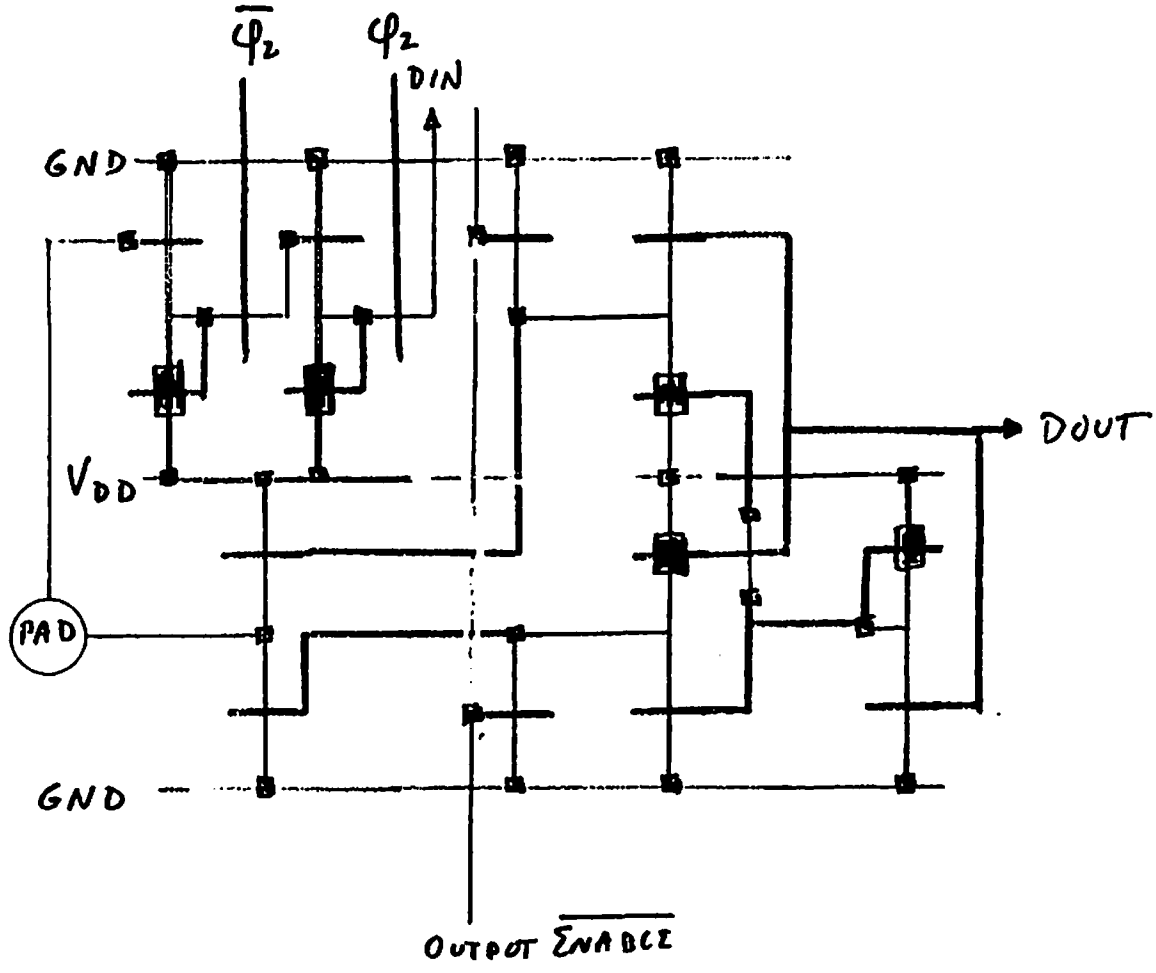


FIG 4.5 TRI-STATE BIDIRECTIONAL PAD CIRCUIT

Figure 4.3 shows one bit of the static registers used here. The output may be gated onto the DOUT bus, and both true and complement outputs are available for use in other parts of the chip. The stick diagram for two bits of the register is shown in Figure 4.6.

A single stage of the static edge-triggered counters is shown likewise in Figures 4.4 and 4.7. Because the chip is being run at the microprocessor clock rate, typically 1000 or 500 nsec, ripple propogation time should not be a problem. Conversely, both the address counter and the iteration counter may be incremented so slowly that a static design seemed necessary. The circuit as shown is a general-purpose JK flip flop which triggers on the rising edge of the clock and has provision for asynchronous clear and preset. When used as a ripple counter, J and K are both grounded and the output from the previous stage is used as the clock signal. Although not used as such in this part of the chip, this circuit can also be easily adapted as a D flip-flop or equipped with a provision for asynchronous load.

A typical layout for the counter decode circuitry is shown in Figure 4.8. The layout has not been completely finalized as yet because optimization of the layout of decoding for the intersection chips may require some additional output lines. After these are added, we expect that some of the input lines will be found to be unnecessary and will be deleted; they have been retained at this time to maintain full generality.

Figure 4.6: Stick figure for two bits of the static registers.

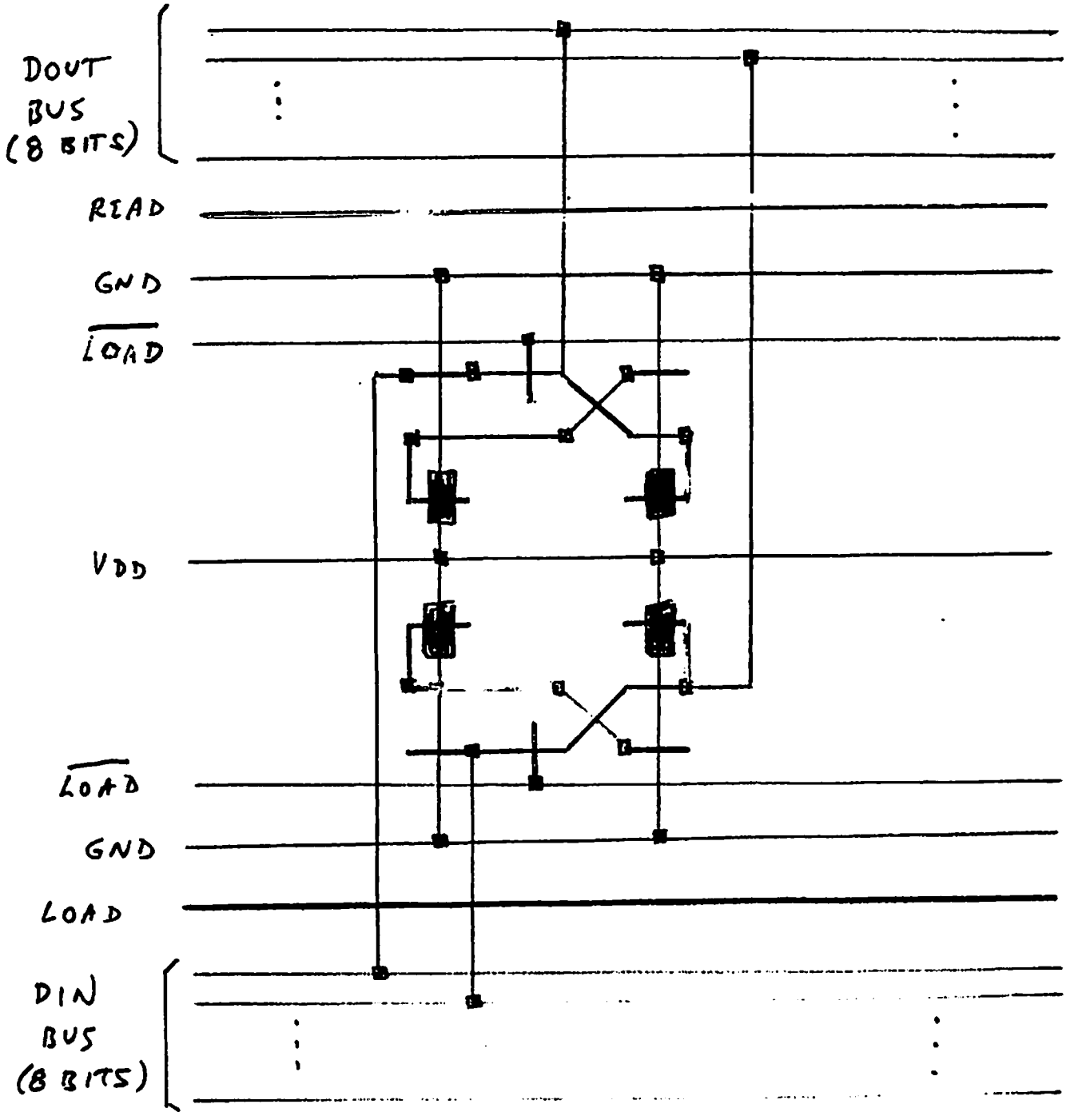
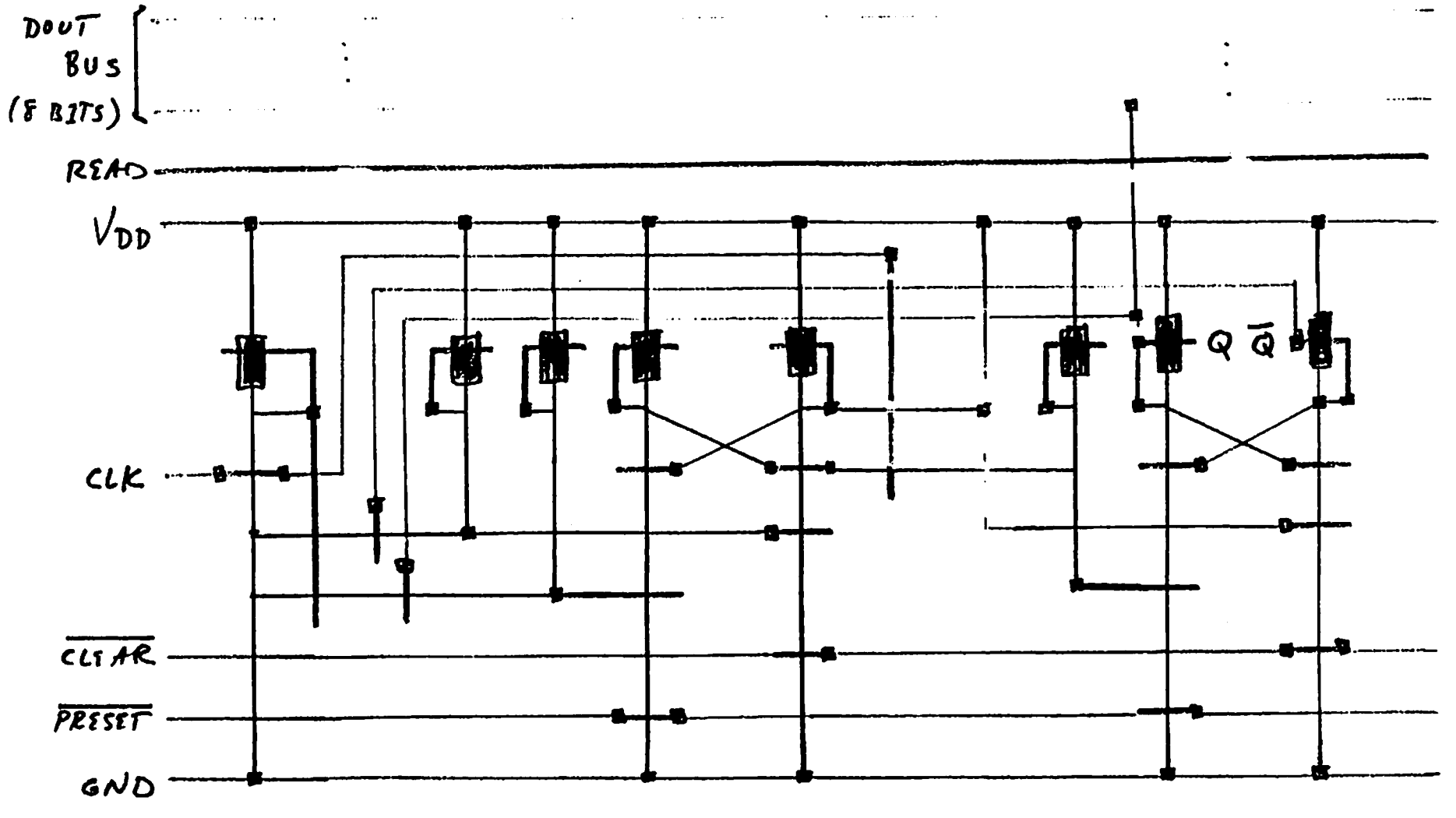


FIG. 4.6 REGISTER (2 BITS SHOWN)

KTL 12/14/81



KTR 12/14/81

FIG. 4.7 STATIC EDGE-TRIGGERED FLIP-FLOP

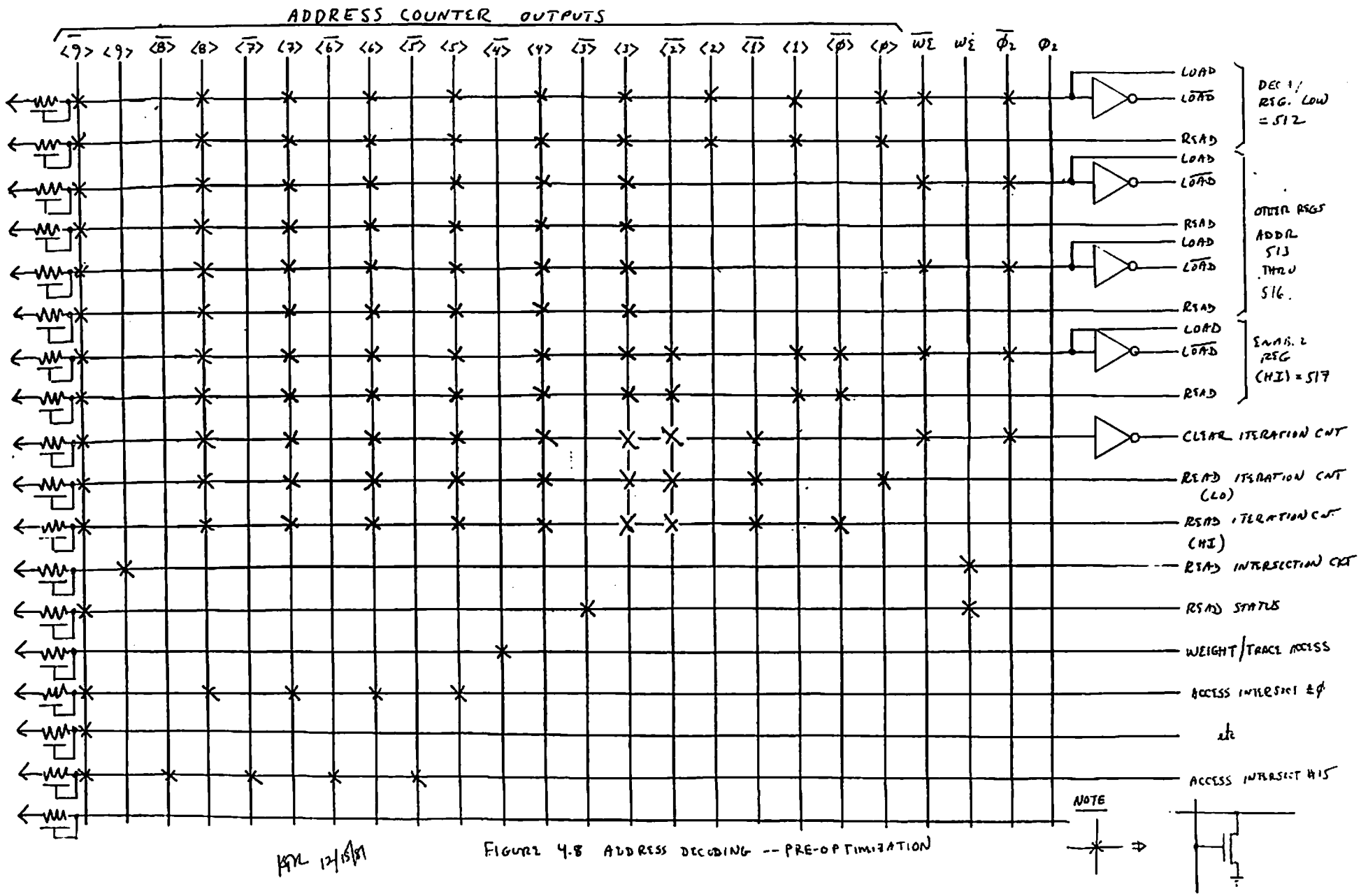


FIGURE 4.8 ADDRESS DECODING -- PRE-OPTIMIZATION

The process of loading data into and reading from the intersection registers can best be understood by examining Figure 4.1 together with 3.1. When the address counter is at any value from 0 through 255 (which can only happen during the PAUSE period), one of the INTERSECTION ACCESS lines shown in Figure 4.8 is asserted (these are the same as the L input select control lines refered to in section 3); the READ INTERSECTION line is also asserted unless it is a write operation (this is the same as the R/W control line refered to in section 3). The intersection registers will shift right once for each pulse on the CHIP SELECT line thus either loading data from the D line (if a write operation) or placing the shifted bit on the D line which connects to the DOUT BUS via the pass transistors shown in Figure 4.1. Whether it is the weight or trace register which is being loaded/examined is controlled by the W_{shift} and T_{shift} lines which are derived from bit 4 of the address counter.

5. TIMING AND CONTROL FUNCTIONS

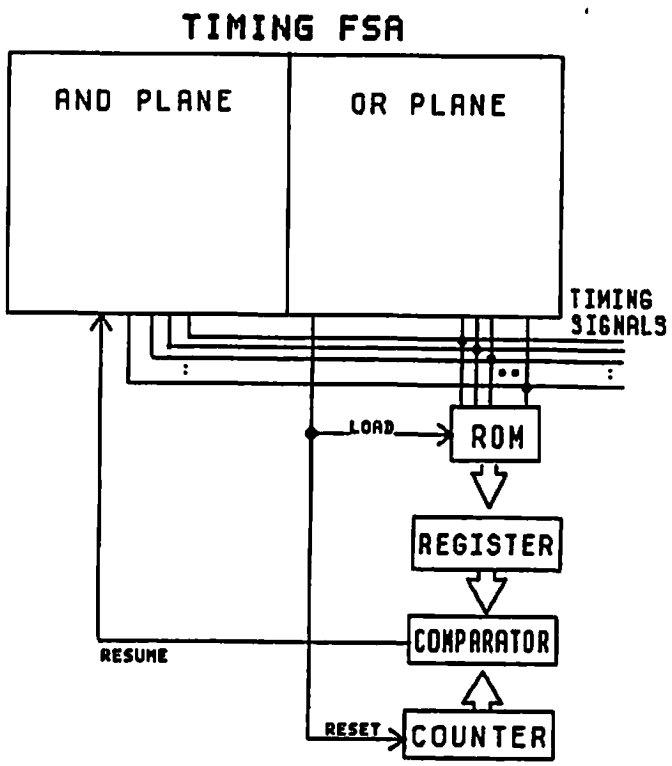
Because the architecture of the ASN chip is so highly pipelined, the control of the parallel operation of the various chip components takes on considerable importance. In order to ensure the integrity of these timing functions, a distinct subtask within the overall design effort was the development of a finite-state machine (FSM) which generates and distributes the timing signals.

Overview

The Timing FSM (see Figure 5.1 and Appendix I for more detail) is implemented as a programmed logic array (PLA). To a first approximation, each clock cycle will see an advance from one state to the next within the PLA. The output lines of the PLA then constitute the timing signals themselves; these are also fed back into the and-plane of the PLA to specify the next state.

Such a scheme is adequate for relatively simple timing requirements, but the internal structure of the PLA will almost certainly be laregly redundant if there are periods during which all timing signals do not change for several clock cycles. The FSM must still advance to a new state at every clock pulse in these cases, but the timing signals do not change; only the additional "overhead" lines

Figure 5.1: The Timing FSA.



running from the or-plane back into the and-plane will change, as they count the clock pulses until the FSM is to assert new timing signals.

It is obvious that this is a rather inefficient use of the PLA, since large portions of the PLA would be devoted simply to counting. The Timing FSM is therefore augmented with a ripple counter which performs this counting operation when all timing lines are to remain in their current state for several clock cycles.

Operation

The FSM timing signal output is fed into a read-only memory (ROM), and serves as a ROM address of a 6-bit "timing word" which determines how long the counter will run.[1] This timing word is latched into a register; when the counter has counted up to the value stored in the register, a comparator asserts the RESUME condition, allowing the PLA to advance to the next state. The loading of the address into the ROM and the resetting of the counter are both performed by the LOAD/RESET line provided by the or-plane of the PLA.[2]

---

1. The counter is actually running (synchronously) at all times, and is simply reinitialized by the RESET line.

2. Note that when LOAD is asserted, the PLA stops advancing state until RESUME is asserted.

The Counter

The counter is implemented as a ripple counter. This is possible because the clock speed for the ASN chip is 1 MHz, which is substantially slower than the ripple time for a 6-bit counter in this technology. Were the chip to run at a very high clock rate, however, it would probably be desirable to substitute some form of parallel counter for the one used here.

The Comparator

The comparator is simply a tree of exclusive-or gates arranged in the obvious fashion, such that RESUME is asserted iff all bits of the register and counter are equal.

The Decay Registers

In order to facilitate the decay of the Trace registers, so that the recent past will be weighted more heavily than the distant past, a special pair of "decay" registers is maintained. The Decay Register is loaded during chip initialization by the UPI (in "pause" mode) with some scalar constant which specifies the number of major clock cycles to pass before the trace is to be "decayed". This value is also loaded into the Decay Count Register, a decrementing counter which asserts its output signal only during the major clock cycle for which it contains zero. This Decay Count Register pulse cues the Trace registers to shift right one bit, effectively dividing by two their current contents and thus decreasing the weight of experience gained prior to the right-shift. The Decay Count Register pulse also signals the Decay Register to reload the Decay Count Register with the decay constant. The contents of the Decay and Decay Count registers can be written or read during "pause" mode for monitoring and testing purposes.

6. SUMMARY

We have presented a plan for the implementation of an associative search network in NMOS VLSI technology. In the course of this work a number of interesting architectural devices have been developed, nearly all of which came about as a direct result of the need for highly pipelined operation. Curiously, this pipelined architecture simplified the design rather than complicating it. The pipelined architecture was viewed as necessary if we were to achieve optimal on-chip space efficiency, but it also fortuitously results in a great speedup in chip operation -- providing, in fact, far more speed than we can take advantage of given the slowness inherent in the problem domain.

An important consideration has been the development of a both on- and off-chip capabilities supporting initialization, monitoring, and reconfiguration of the chip. The glamour of VLSI design often seems to overshadow the less exciting but equally important development of appropriate testing approaches and equipment; aware of this hazard, we invested a great deal of time in the design of a microprocessor interface capable of rigorously exercising the chip and providing it with a means of talking to the real (or at least research) world. This effort paid off during the design of the chip as well, in that the establishment of protocols for use of the chip by an experimenter made us aware of several subtleties in design that we might have missed had we merely been planning another piece of what industrial VLSI designers

humorously refer to as "silicon jewelry".

Finally, we are compelled to note that this design document, for all its bulk, is still a strategy rather than a specification. We are only too well aware of the fact that considerable work remains before the chip will be ready for the fabrication line, most notably in further layout work and in the grueling but critically important domain of low-level circuit simulation. Furthermore, we have no illusions as to the utility of the chip described here in the context of, for example, industrial application. Associative search networks themselves constitute a fertile field of research, and we look forward to improvements in ASN technology that will make future VLSI ASN chips not only interesting and instructive, but of practical value as well.

References

[1] Amari, S. A mathematical approach to neural systems. In: Systems Neuroscience. Metzler, J., ed. New York: Academic Press, 1977.

[2] Anderson, J. A., Silverstein, J. W., Ritz, S. A., Jones, R. S. Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psychological Review, 1977, 85, 413-451.

[3] Barto, A. G., Sutton, R. S. Landmark learning: An illustration of associative search. To appear in Biological Cybernetics, 1981b.

[4] Barto, A. G., Sutton, R. S., Brouwer, P. Associative search network: A reinforcement learning associative memory. Biological Cybernetics, 1981, 40, 201-211.

[5] Bergland, G. D., A Guided Tour of Program Design Methodologies. Computer, 18:13-37 (Oct. 1981).

[6] Duda, R. O. and Hart, P. E., Pattern Classification and Scene Analysis. New York: Wiley (1973).

[7] Hinton, G. E. Implementing associative networks in parallel hardware. In: Hinton and Anderson, Parallel models of associative memory. Lawrence Erlbaum Assoc., Hilsdale, N.J. (1981).

[8] Hinton, G.E., and Anderson, J.A. (eds.) Parallel models of associative memory. Lawrence Erlbaum Assoc., Hilsdale, N.J. (1981).

[9] Kohonen, T. Associative memory: A system theoretic approach. Berlin: Springer, 1977.

[10] Longuet-Higgins, H. C. Holographic model of temporal recall. Nature, 1968, 217, 104.

[11] Mead, C., and Conway, L. Introduction to VLSI systems. Addison-Wesley, Reading, MA, 1980

[12] Nakano, K. Associatron - a model of associative memory. IEEE Transactions on Systems, Man, Cybernetics, 1972, SMC-2, 3, 380-388.

[13] Narendra, K.J. and Thathachar, M.A.L. Learning automata — a survey. IEEE Trans. Sys., Man. Cyber. SMC-4, 4:323-334. (1974).

[14] Palm G. On associative memory. Biological Cybernetics, 1980, 36, 19-31.

[15] Rosenblatt, F. The Perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. 65:386-408, 1958.

[16] Tsetlin, M. L. Automaton theory and modeling of biological systems. New York: Academic Press, 1973.

[17] Widrow, B. A statistical theory of adaptation. In: Adaptive Control Systems, edited by F. P. Caruthers and H. Levenstein. Pergamon Press, New York.

[18] Wood, C. C. Variations on a theme by Lashley: Lesion experiments on the neural model of Anderson, Silverstein, Ritz, and Jones. Psychological Review, 1978, 85, 582-591.

A P P E N D I X A

TIMING OF SIGNALS AND OPERATIONS IN ITERATE MODE

This appendix describes at a detailed level (the level of single clock cycles) the timing of the various aspects of the network's operation when it is in iterate (normal) mode, including specification of what control and data signals are necessary on each input line at each time. Figure 1.3 provides an overview of the timing for Iterate mode.

The overall Iterate step is between 45 and 61 cycles long, depending on the value of the Nc parameter, which is related to c in equation 3. Extensive reference is made in the following text to the control and data signals of the intersection circuits shown in figures 3.1 and 3.4 and discussed in Section 3.3.

Weight Calculation

cycle 0: The first cycle is spent checking the pause enable bit of the status register. Only if this is 0 does the iterate phase continue.

cycle 0: The shift-adder is assumed to be empty (all 0s). Crucial control lines : Tshift=0, Wshift=0, Xc=0, Rc=0.

cycle 1: The multiplication of the weight registers by the X-input begins. The X-input starts appearing bit-serial on each Xc, and the products $w_{ij} x_i$ start appearing bit-serial as the output of each intersection component on the WX output line. Crucial control lines: Rc=0, Wshift=0, SignExtend=1.

cycle 5: After 4 cycles, the X-inputs have been fully read in bit serial on the Xc input lines. Xc becomes 0. SignExtend becomes 0. Output bits of the products continue to appear on the WX output lines.

cycle 20: After 16 cycles, the last bit of each product has been generated and output on the WX output lines of each intersection circuit.

Trace Calculation

cycle 1: The update of the trace registers also begins now. The D data line of each network element is set to the previously computed value of its Y output. The 4-bit X-input saved from the previous time step begins to appear bit-serially on the Xold input lines to each intersection circuit. (Note that the new value of the X-inputs is simultaneously being saved in the input buffer as this old value is being read out -- the input buffer is a simple 4-bit shift register.) The intersection circuits begin to AND these two signals to form the product of Yold and Xold, and to add this product bit-serially to the contents of the trace register as the bits in the register shift out of the LSB and are circulated around into the MSB. Crucial control lines: Tshift=1, R/W=1, L=0.

cycle 5: After 4 cycles, the old values of the X-inputs have been fully read in. The Xold inputs to the intersection circuits become zero. The remaining 12 bits of each trace register continue to circulate until they are back into their proper places.

cycle 16: 16 cycles after the beginning of the trace update period, the trace registers have had Yold times Xold added to them. If the Decay-Count register is still non-zero, then it is decremented by one and the Tshift control lines are set immediately to zero. If the Decay-Count register has fallen to zero, then the trace register must be divided by two. In this case, Tshift is held at 1 for an additional cycle, with D=0, L=1, and R/W=0.

Sum Calculation

cycle 1: The products appearing on the WX output line of each intersection circuit begin to be combined by the adder tree (see Figure 2.1) to yield the element sums s_i.

cycle 5: The first bits of the sums s_i become available at this time. Noise addition and the thresholding operation begin at this time.

cycle 24: The last bit of the sums s_i are available at this time. Noise addition and thresholding is complete. The Y-outputs have been computed and are available.

Z Calculation

cycle 1: z is read in bit-serial and stored in the z shift register buffer.

cycle 20: The updating of the weight register begins at this time (immediately after the product $w_{ij} x_j$ leaves the shift-adder). Z is then read bit-serially out of the z shift register and into the z_{old} shift register. Meanwhile, the previous value of z, currently in the z_{old} shift register, starts being shifted out. As this shifting is done, the difference between z and z_{old} is computed and fed bit-serial into the intersection circuits on their Rc input lines. The first bit of this difference (also called Dz) appears at this time and the intersection circuits begin multiplying the value of the trace register by this Dz input. The product Dz $T_{ij}(t)$ starts appearing bit-serial on the WX output line of the intersection circuit (and is lost - ignored). Crucial control lines: SignExtend=1, Wshift=0, Tshift=0.

cycle 28: 8 cycles later, Dz has been completely read in and Rc becomes 0. SignExtend becomes 0.

cycle 28+Nc: Nc cycles later (where Nc is determined by c) we start the actual adding into the weight register. Crucial control lines: Wshift=1, R/W=1, L=0.

cycle 28+Nc+16: 16 cycles later, the weight register is fully updated, and the shift-adder is empty. Wshift becomes zero. The iterate step has been completed.